A Layout Estimation Algorithm for RTL Datapaths†

Mehrdad Nourani and Christos Papachristou

Department of Computer Engineering
Case Western Reserve University
Cleveland, OH 44106 USA

Abstract— This paper presents a fast and efficient algorithm to estimate the area cost of a given RTL datapath. This is achieved by considering the physical length of components (provided by a component library) and connections data (given by the datapath description) within an actual layout model and using analytical formulas in a constructive algorithm. Our layout estimator uses a non-probabilistic basis to estimate the overall cost of a datapath structure. It can estimate the overall layout with emphasis on area minimization, or can be adapted to delay minimization depending on the user request. Layout area is estimated for Standard-Cell and Full-custom layout methodologies at the RT level.

I. Introduction

A. Motivation

High level synthesis has emerged as an important discipline for designing register-transfer level (RTL) circuits from behavioral specifications. There are two important processes in high level synthesis: *operation scheduling* and *resource allocation*. The overall synthesis goal is to produce an RTL circuit, i.e. datapath and controller satisfying the behavior and subject to design constraints. The actual circuit layout can be generated from the datapath using a silicon compiler.

The question of constraints is crucial for the ultimate success of high level synthesis. Scheduling and allocation are both NP-complete problems. There are several good scheduling and allocation methods which handle technology independent constraints very well, i.e. optimize datapaths with respect to the number of ALUs, registers and multiplexers. However, these optimizations may not satisfy constraints at the actual chip layout level. To address this deficiency, recent synthesis techniques try to optimize with respect to the area rather than the number of datapath components, using library-based component cost [1][2]. Although these designs are more realistic, they are non-optimal because they are usually based on heuristics. More importantly, these techniques still do not consider placement and routing of components which significantly affect the overall chip area. There are some attempts to alleviate this problem during the RTL design process using stochastic layout models, however, the accuracy of these models is not known [3][4][5].

In this work we propose a fast layout estimation process which is a bridge between high and low level in the design process. The estimator can be iteratively invoked in higher levels (e.g. scheduling and allocation) to provide some guides which finally improve RTL datapath and its overall layout cost. The motivation for this approach is twofold. First, because of the non-optimal nature of the synthesis process, implementing a fast and reasonably accurate *predictor* (for area or time) creates a stronger guiding mechanism to search the design space and consequently improve our decisions in the design process. Second, a layout estimator based on a realistic datapath layout model can quickly estimate the complete layout cost (the components and routing cost) in the course of design modifications. It should be noted that in this work we do not consider the controller area.

B. Related Works

The layout estimation algorithms aim at developing tools to facilitate and guide the circuit design task by providing means for the designers to a priori evaluate the final chip layout of the circuit. Such tools are necessary due to the lengthy and time consuming nature of the design tasks, such as placement, floor planning and layout synthesis [6][7][8], and would significantly contribute to shortening the design turn around time. Furthermore, a layout estimator is needed to automatically estimate realistic module aspect ratio for floor planning in order to reduce not only the overall design cost, but also the designer's effort to complete an actual chip layout. The existing estimation approaches can be classified into two broad categories: *constructive* and *analytical* [3] [4].

In constructive methods, the design description is repeatedly partitioned using a slicing tree. Some of the generic partitioning algorithms have been described in [9]. The slicing tree represents the connections between different partitions. The process recursively proceeds to the leaves of this tree which represent the cells of a given library. Then, the cells are combined up to the root level to generate the complete chip. Constructive methods are slow but more accurate than analytical models [3][10].

In analytical methods, the major effort is to estimate the wiring space and the relative position of the cells using some analytical formulas. Almost all the wiring models proposed so far are stochastic in nature. Nondeterministic methods are more attractive because the empirical relations existing in well-placed designs validate the probabilistic models for wire distribution [11]. Furthermore, dealing with a distribution function to estimate routing area is much easier than considering all point-to-point wires. The drawback, however, is restriction of design space and less accurate results. Work presented in [12] uses a mixed analytical-constructive model. An analytical model predicts the cost of leaf subcircuits and then a constructive model computes the cost of the entire layout design by traversing the slicing tree bottom up.

C. Problem Description and Paper Organization

Given an RTL description, i.e. datapath description, how can we estimate the dimension of the physical layout

†This work was partially supported by the Semiconductor Research Corporation (SRC) under contract No. 92-DJ-147.

of that circuit on a chip without actually performing the layout task? An estimator which is accurate to within 10% of the actual area produced by a commercial silicon compiler is considered *very good.*

Section II reviews our layout model. In section III, we present the estimation algorithm by means of a small size example chosen from the literature. Section IV extends the algorithm to include full-custom (non-standard) cell and time consideration. The experimental results are shown in section V. Finally, concluding remarks are in section VI.

II. Estimation Level and Layout Model

The information about components and connectivities may be constructed in different level of design hierarchy, such as RT level, logic block level or even gate level. RT level is our preferred level to apply the estimation process since the components in this level (such as ALUs, multiplexers and registers) are very well-known and accessible in most libraries. Also, the number of components which we must connect is relatively small and so the overall process will be faster. To clarify this, consider the datapath of the *second order differential equation solver (HAL)* [13] shown in Fig. 1. If we have the ALUs, multiplexers and registers in the library, the estimator considers 22 components (4 ALUs, 7 MUXs and 11 REGs) in all. At the logic block level, we should distinguish other subcells for ALUs, MUXs and REGs. For example, a 2×1 multiplexer is a combination of two AND gates, one OR gates and an inverter. This increases the number of components drastically and the estimation process would be slower.

To make the process even faster, the user may define another level for layout estimation based on the regularity of the RTL structure observed. For example, in the datapath of Fig. 1, the user may benefit from the regularity of the given datapath by defining a combination of an ALU, two input multiplexers and the output registers as a module M^i. We have specified these modules, $M^1, \cdots, M^4$, by dashed lines in Fig. 1. Using the new modules, the estimator considers only 11 components (4 defined modules and 7 registers). However, the user should introduce the new cells, $M^1, \cdots, M^4$, into the library. Fortunately, in our regular datapath, because of the unique connection between MUX-ALU and ALU-REG, each module ($M^i, 1 \leq i \leq 4$) can be trivially constructed from components in the conventional libraries. However, our estimator is not restricted to these modules only.

A typical cell in the library is shown in Fig. 2. The following are the assumptions and characteristics of that model:

- All cells have the same height h_c. This implies that we have only standard cells, but later we will relax this assumption to generalize our method to any arbitrary size cell.
- Cells are arranged in rows of roughly equal size based on the *aspect ratio* requested by the user.
- Two-layer HV (Horizontal-Vertical) routing model is considered for computing the wire lengths. The analytical formulas estimating wire lengths should be modified for more general routing models.
- All pieces of information shown in Fig. 2 should be provided prior to invoking the estimator. W and H are the total width and total height of the cell, respectively. h_a and h_b represent the routing channel above and below the cell. Assume a fixed orientation for the cell and consider that the left bottom corner is located at the origin of a Cartesian system. Since dealing with input and output coordinates of all points is difficult, we use one coordinate for input and output points, denoted by P_{in} and P_{out}, respectively.

  Assume $I_1, I_2, \cdots, I_r$ and $O_1, O_2, \cdots, O_s$ represent x-coordinates of input and output points of the cell. We locate P_{in} and P_{out} such that over and under estimations for wire lengths cancel each other: $P_{in} = 1/r * \sum_{i=1}^{r} I_i$ and $P_{out} = 1/s * \sum_{i=1}^{s} O_i$.

III. Layout Estimation Process

We will explain the different steps by using a simple datapath which is actually a part of HAL benchmark [13]. Fig. 1(a) shows this datapath, but as we said earlier we consider a part of it for simplicity. We assume that the datapath consists of four module M^1, M^2,M^3 and M^4. As mentioned earlier, each module corresponds to the combination of an ALU and its two multiplexers specified by dashed lines in the figure. Fig. 1(b) provides corresponding schedule and interconnect details. For example, in control step 2, M^2 and M^3 send two signals, $mul1$ and $mul0$, respectively, to M^4 through different wires. M^4 sends $mul2$ and $mul4$ through different wires to M^1 and M^2, respectively.

After reading the datapath and the library cells, the estimation algorithm performs two phases. *Phase 1* places all modules in one row and *Phase 2* folds the one row to obtain a multi row design to satisfy an *aspect ratio.* These two phases can be seen in most of the classical placement or estimation methods such as [3][5]. However, our estimator is different in three aspects. First, we do not use any probabilistic method to estimate the wire lengths. Instead, based on the information about input/output ports of a cell (provided by cell library), we estimate them by a more realistic process. Second, by using a flexible layout model for components, we do not need to consider feedthroughs (splitting cells to route wires to cells which are more than one row apart), and thus less computations and faster runtime. Third, we can perform layout estimation by considering area minimization (i.e. area-driven) or delay minimization (time-driven) with the same model. This is important in real experiments to predict the results based on designer's choice.

A. Phase 1

In this phase a one-row design is constructed in the following five steps:

- <u>*Step 1*</u>: Construct the connection matrix.

$$c_{ij} = \begin{cases} \text{Number of wires}^* \text{ through which} & \text{if } i \neq j \\ M^i \text{ sends signals to } M^j & \\ 0 & \text{if } i = j \end{cases}$$

  $c_{ii} = 0$ shows that we are assuming that intra-module connections do not impose any area overhead to the design layout since they can be routed over the cell using different layers of metal. For our example the matrix is:

$$C = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

TABLE I

CONNECTIVITY FACTORS FOR THE RUNNING EXAMPLE

Module i	S^i	R^i	f^i
M^1	1	2	3
M^2	1	2	3
M^3	2	0	2
M^4	2	2	4

- *Step 2*: Select two heavily connected modules.

  Let us assume there are n modules ($n = 4$ in our example) in the datapath. $S^i = \sum_{j=1}^{n} c_{ij}$ shows the number of wires through which module M^i sends signals to all other modules in the datapath. Similarly, $R^i = \sum_{j=1}^{n} c_{ji}$ shows the number of wires through which module M^i receives signals from all other modules in the datapath.

  We define the *connectivity factor* as follows:

  $$f^i = S^i + R^i = \sum_{j=1}^{n}(c_{ij} + c_{ji})$$

  The connectivity factors in our example are tabulated in Table I.

  The module with the largest connectivity factor is defined as the *first heavily connected module (FHCM)*. Intuitively speaking, a module which has the largest number of wires to send (receive) signals to (from) all other modules is the first heavily connected module. Formally:

  $$M^{i^*} \ is \ FHCM \Leftrightarrow f^{i^*} = \overset{1 \leq i \leq n}{MAX} \{f^i\}$$

  In our example M^4 with connectivity factor $f^4 = 4$ is FHCM.

  The *second heavily connected module (SHCM)* is defined as a module which has the most connections with respect to FHCM. It is not hard to see that SHCM can be obtained from the following relation:

  $$M^{j^*} \ is \ SHCM \Leftrightarrow t_{i^*j^*} = \overset{1 \leq j \leq n}{MAX} \{t_{i^*j}\}$$

  where $t_{ij} = c_{ij} + c_{ji}$. So, $j^* = 2$ since t_{42} is the largest element in the fourth row. This means that M^2 is SHCM in our example.

- *Step 3*: Choose a configuration for the selected modules.

  Considering the possible orientations and permutations, we have eight (2^3) possible configurations to make a new module of two cells. Four of these eight configurations are mirror images of the rest with respect to the vertical axis and need not to be considered since the results are identical. Fig. 4 shows the two orientations for a module while Fig. 3 shows the four non-identical configurations. The following four relations compute the total wire length of all connections between the two selected modules for the four configurations, respectively:

  $$L_1^{ij} = c_{ij}(W^i + P_{in}^j - P_{out}^i) + c_{ji}(W^i + P_{out}^j - P_{in}^i)$$
  $$L_2^{ij} = c_{ij}(W^i + Q_{in}^j - P_{out}^i) + c_{ji}(W^i + Q_{out}^j - P_{in}^i)$$
  $$L_3^{ij} = c_{ij}(W^j + P_{out}^i - P_{in}^j) + c_{ji}(W^j + P_{in}^i - P_{out}^j)$$
  $$L_4^{ij} = c_{ij}(W^j + P_{out}^i - Q_{in}^j) + c_{ji}(W^j + P_{in}^i - Q_{out}^j)$$

  where $Q_{in}^j = W^j - P_{in}^j$ and $Q_{out}^j = W^j - P_{out}^j$. We choose the configuration k^* such that:

  $$L_{k^*}^{ij} = \overset{1 \leq k \leq 4}{MIN} \{L_k^{ij}\}$$

- *Step 4*: Make a composed module.

  After determining the best configuration, we now construct a new module consisting of the two selected modules. The new module corresponds to a composed cell in the library of the same type. We must also compute the required information about it. Let M^i and M^j be two selected modules and $M^{i \cup j}$ be the new composed cell (Fig. 5). We can extract the necessary information as follows:

  - $h_a^{i \cup j} = h_t^{i \cup j} + MAX\{h_a^i, h_a^j\}$
  - $h_b^{i \cup j} = MAX\{h_b^i, h_b^j\}$
  - $H^{i \cup j} = h_c + h_a^{i \cup j} + h_b^{i \cup j}$
  - $W^{i \cup j} = W^i + W^j$

  where $h_t^{i \cup j}$ is the channel height for passing the connections between M^i and M^j. We saw that $t_{ij} = c_{ij} + c_{ji}$ shows the number of tracks in this channel. We need two layout technology-dependent factors: minimum thickness of tracks and minimum distance between tracks, denoted by a and d, respectively. So, we have: $h_t^{i \cup j} = t_{ij}(a + d)$. $P_{in}^{i \cup j}$ and $P_{out}^{i \cup j}$ are also computed based on P_{in}^i, P_{in}^j, P_{out}^i and P_{out}^j in a similar way that we computed P_{in}^i or P_{out}^i in section II.

- *Step 5*: Update the connection matrix.

  We replace $M^{i \cup j}$ into M^i and M^j and update connectivities by manipulating rows and columns of C corresponding to the selected modules. To update the connectivity matrix C^{old}, we first add the ith and jth columns of C^{old} to find total connections required for incoming signals from all modules to $M^{i \cup j}$ (matrix C'). Then we add the ith and jth rows to find similar numbers for outgoing signals to be sent from $M^{i \cup j}$ to other modules (matrix C''). Finally, we set the diagonal element corresponding to the new module to zero since the connections of a module to itself have been already considered during the composition of $M^{i \cup j}$ (matrix C^{new}). In our example:

  $$C^{old} = \begin{pmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 \end{pmatrix} \Rightarrow C' = \begin{pmatrix} 0 & 0 & 1 \\ 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 1 \end{pmatrix} \Rightarrow$$
  $$C'' = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 2 \end{pmatrix} \Rightarrow C^{new} = \begin{pmatrix} 0 & 0 & 1 \\ 1 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

  In this example, the third column in C' (corresponding to $M^{i \cup j}$) is the summation of the second and fourth column of C (corresponding to the selected modules M^2 and M^4). Similarly, the third row in C'' is the summation of the second and fourth row of C'. Finally, matrix C^{new} is matrix C'' in which the element corresponding to $M^{i \cup j}$ is set to zero.

*These are really *bus-type* wires at RTL design. For example, in an 8-bit design each wire is an 8-bit width bus.

If C is 1×1 matrix (meaning all modules have been merged and one row design has been constructed) go to *Phase 2*, otherwise return to step 2.

At the end of the first phase, we have the placement of modules, their orientations and the total width (W_1) and total height (H_1) of the one row design. Fig. 7(a) shows the one-row result for our example. Note that for brevity we did not show the estimator decisions in each step.

B. Phase 2

The estimator folds the one row design to obtain a 2-dimensional practical size layout. Fig. 7(b) shows the design after folding. The area shaded by dashed lines in this figure shows the overhead caused by breaking the one row design. W^*, H^* and N^* are total height, width and the number of rows of the multi row design after folding, respectively. Assuming $d \ll W^*$ we have three relations between the one row design and the multi row design:

$$\begin{cases} H^* W^* = H_1 W_1 \\ \frac{H^*}{W^*} = A_r \\ H^* = N^* H_1 \end{cases}$$

where A_r is the given aspect ratio. *Phase 2* proceeds in the following three steps:

- Step 1: Compute the ideal folding parameters.
  By solving the above equations for H^*, W^* and N^* we have:
  $$\begin{cases} H^* = \sqrt{H_1 W_1 A_r} \\ W^* = \sqrt{\frac{H_1 W_1}{A_r}} \\ N^* = \lceil \sqrt{\frac{W_1 A_r}{H_1}} \rceil \end{cases}$$
  Set $x = 0$ and $i = 0$. x is a variable showing the incremental length of one row design folded. When x exceeds W_1 we stop the folding process. i is a variable showing the total number of rows created by folding.
- Step 2: Construct a row after folding.
  - Compute next iteration $i = i + 1$ and construct $row[i]$ as the portion between x and $x + W^*$ in the one row design.
  - We need to compute the interconnection cost overhead after breaking the one row design. This overhead is proportional to the number of broken tracks in the routing channel above the modules. This number, denoted by N_b, depends on the orientation and and the relative position of a module in the constructed one row design. Fig. 6 shows the position of M^i among other modules in the one row design. Assuming that x shows the left coordinate of M^i (a typical module) in the one row design, we distinguish three regions, such that in each region N_b remains the same. Let S_{left} and S_{right} be the set of modules in left and right hand side of M^i in the one row design, respectively. Table II shows the number of tracks to be broken depending on the region at which breaking occurs. The interconnection area overhead after this breaking is proportional to the increase of row width by $N_b(a + d)$. Recall that a and d are two layout technology-dependent factors: minimum thickness of tracks and minimum distance between tracks, respectively. Other parameters in this table are:

TABLE II
COMPUTING N_b FOR THREE REGIONS ABOVE MODULE M^i

Regions	*Range*	N_b
Region 1	$w \in [x, x + P^i_{in}]$	$n_1 + n_3 + n_5 + n_6$
Region 2	$w \in [x + P^i_{in}, x + P^i_{out}]$	$n_2 + n_3 + n_5 + n_6$
Region 3	$w \in [x + P^i_{out}, x + W^i]$	$n_2 + n_4 + n_5 + n_6$

  $n_1 = \sum_{l=1}^{n} c_{li}$ *where* $M^l \in S_{left}$, $n_2 = R^i - n_1$,
  $n_3 = \sum_{l=1}^{n} c_{il}$ *where* $M^l \in S_{right}$, $n_4 = S^i - n_3$,
  $n_5 = \sum_{l \in S_{left}} (\sum_{r \in S_{right}} c_{lr})$ and
  $n_6 = \sum_{l \in S_{right}} (\sum_{r \in S_{left}} c_{rl})$.
  - Find the height and width of this row denoted by $row[i].height$ and $row[i].width$.
  - Compute $x = x + W^*$. If $x < W_1$, repeat step 2 otherwise go to step 3.
- Step 3: Compute the final dimensions of the design.
  After breaking the one row design into equal size rows, we can compute number of rows, height and width of 2-dimensional layout as follows:
  $$\begin{cases} N_2 = i \\ H_2 = \sum_{j=1}^{N_2} row[i].height + (N_2 - 1) * d \\ W_2 = \underset{1 \le j \le N_2}{MAX} \{row[j].width\} \end{cases}$$

C. Algorithm

The summarized estimator algorithm in pseudocode, and its running time analysis are presented below.

```
MakeOneRowDesign()
{
      ReadLibraryCell();
      ReadDataPath();
      size=number_of_modules_in_datapath;
      ConstructConnectionMatrix();
      while (size > 1)
      {
            SelectTwoHeavilyConnectedModules();
            SelectTwoOrientations();
            MakeComposedModule();
            UpdateConnectionMatrix();
            size = size - 1;
      }
      H1=height_of_one_row_design;
      W1=width_of_one_row_design;
}
FoldOneRowDesign()
{
      ComputeIdealFolding(H*, W*, N*);
      x = 0.0
      i = 0;
      while (x < W1)
      {
            i=i+1;
            FindModuleToBreakAt(x + W*);
            ConstructRow(i);
            x=x+row[i].original_width;
      }
      N2 = i;
      H2 = SUM_{j=1}^{N2} row[i].height + (N2 - 1) * d;
      W2 = MAX_{1<=j<=N2} {row[j].width};
}
```

The outer *while* loop in the first phase (making one row design) repeats exactly $n-1$ times where n is the total number of modules in the datapath. All functions in the body of this *while* loop are executed at most in $O(n)$ since in the worst case we need to check all n modules to select two of them or to make the composed module. Thus, the running time of this phase is $O(n^2)$.

Phase 2 (folding) runs in $O(N^*) = O(\lceil\sqrt{\frac{W_1 A_r}{H_1}}\rceil)$ since the *while* loop is repeated approximately N^* times. All factors W_1, H_1 and A_r are constants and all functions in the body of the loop use analytical formula with no selection, sorting and so on. So, we conclude that this time is $O(c)$, a constant time. Hence, the overall running time is $O(n^2)$. The estimator is fast enough to be effectively used within the datapath design process.

IV. Extensions

In this section, we address two practical applications to the synthesis problem: considering the non-standard cells and estimation with emphasising on delay optimization instead of area optimization. The first one is important since a designer usually uses some non-standard cells included in the library. The second one gives the designer the ability to estimate the layout when searching for a fast design.

A. Full-Custom Cells

Designers usually use a library containing some hand-made cells with multi-row configuration. In this section, we address how the estimation algorithm considers the non-standard cells. We assume that each row of a non-standard cell consists of a combination of standard cells. Depending on the nature of a non-standard cell, we have two ways to handle it:

- *Unfolding:* If the size of the non-standard cell is not very large and the connections between rows are not very complex, the user can easily unfold it and make an *artificial unfolded* standard cell, compute the information about the unfolded cell and run the estimator.
- *Row Isolation:* If the size of the non-standard cell is large, or there are complex connections between different rows, unfolding imposes a very rough estimation of that cell since fast computation of the necessary information about the unfolded cell (e.g. P_{in}, P_{out}, h_a, h_b and so on) degrades accuracy. To get a more accurate result, we *isolate* each row (in a non-standard cell) from each other and consider each row as a new *dummy* standard cell. The user, then, provides the necessary information about the new dummy cells which is basically the height and width of each row and the number of connections among them. The drawback, however, is a slower process since the number of modules will be increased by the number of rows in that non-standard cell.

B. Estimation With Emphasis on Delay Minimization

When we described our estimator, we pointed out that we estimate the layout by emphasising area minimization. In that process, we try to find the best orientations and relative positions to save area. However, one of the advantages of our layout estimator over the existing ones is its ability to estimate the layout while considering the overall delay of the datapath.

The existing tradeoff between time and area prevents the estimator from considering both issues simultaneously, however, the estimation algorithm can be adapted to consider the overall delay of datapath as another option. To do this, note that in a *synchronous* digital design, execution time of components, usually expressed by number of control steps (clock cycles), determine the lower bound on clock period. Thus, we are concerned with fast designs (fast clock period) and consequently with interconnection delay optimization. To incorporate interconnect delay within our estimation algorithm, we choose two modules whose connections create the largest delay in the datapath (instead of two heavily connected modules as before) and then apply the same placement process. *First and second heavily activated modules*, denoted by *FHAM* and *SHAM* can be determined using another matrix called the *activity matrix* (A) (similar to C) defined as follows:

$$a_{ij} = \begin{cases} \text{Number of times that } M^i \rightarrow M^j & \text{if } i \neq j \\ \text{path is activated} & \\ 0 & \text{if } i = j \end{cases}$$

Again, we define $a_{ii} = 0$ since we assume that the intra-connection delay within module M^i is negligible compared to the inter-connection delay. The connectivity factor is now defined as:

$$f^i = S^i + R^i = \sum_{j=1}^{n} a_{ij} + \sum_{j=1}^{n} a_{ji} = \sum_{j=1}^{n} (a_{ij} + a_{ji})$$

Considering the datapath description of the example of Fig. 1(a), the activity matrix is:

$$A = \begin{pmatrix} 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 2 \\ 1 & 1 & 0 & 0 \end{pmatrix}$$

For example, as Fig. 1 shows, M^1 sends two values (*add0* and *sub0*) to M^2 meaning the path $M^1 \rightarrow M^2$ is activated twice or $a_{12} = 2$. Similarly, $a_{21} = 0$ since M^2 does not send any data (signal) to M^1. The largest $a_{ij} + a_{ji}$ in this matrix shows the modules and the path for which activation in two directions creates the largest delay in each step of estimation. Formally,

$$M^{i^*} \; is \; FHAM \Leftrightarrow f^{i^*} = \overset{1 \leq i \leq n}{MAX} \{f^i\}$$

$$M^{j^*} \; is \; SHAM \Leftrightarrow t_{i^* j^*} = \overset{1 \leq j \leq n}{MAX} \{t_{i^* j}\}$$

where $t_{ij} = a_{ij} + a_{ji}$. Matrix A will be updated in a way similar to that described for C. Basically, we use the same process when we consider delay minimization. The only difference is that matrix A (instead of C) decides for the selection of two modules in one row construction.

V. Experimental Results

The estimation algorithm described in the previous sections has been implemented in C on a SUN SPARC-IPC workstation. The layout estimation results for different experiments are tabulated in Table III and IV . The CPU time is less than 0.5 seconds for these examples. The

TABLE III
LAYOUT ESTIMATION OF THREE FUNCTIONAL UNITS (AT LOGIC BLOCK LEVEL)

Ex. #	*# of Trans.*	*Actual*		*Estimated*		*Relative Error %*
		Height	*Width*	*Height*	*Width*	
1	521	711	640	733	667	+7.4%
2	2097	1791	1368	1875	1462	+11.9%
3	220	646	534	1068	308	−4.6%

TABLE IV
LAYOUT ESTIMATION OF FOUR DATAPATHS (AT RT LEVEL)

Ex. #	*# of Comp.*	*Actual*		*Estimated*		*Relative Error %*
		Height	*Width*	*Height*	*Width*	
4	24	1747	1987	1431	2614	+7.7%
5	34	5596	4716	4176	5740	−9.2%
6	28	2950	2501	2520	3228	+10.2%
7	32	5419	5595	5265	6365	+10.5%

wall clock time is about 3 seconds. All physical dimensions are expressed in terms of $\lambda^\dagger$ based on the SCMOS standard cell library within the Berkeley OCTTOOLS 5.1 [14]. Note that we have described all examples in BDS (hardware design language in OCTTOOLS). Then, OCTTOOLS, as a silicon compiler, generated the layout automatically. OCTTOOLS needs from a few minutes (for Design 1) up to 8 hours (for Design 7) to generate the complete layout.

Table III consists of three examples for which the estimator predicts overall cost at the logic block level. The major primitive cells in the library for these examples are *Inverter, AND, OR, XOR and Full Adder.* Design 1 and Design 2 are 4-bit and 8-bit multipliers, respectively. Design 3 is a 4-bit ALU capable of performing addition, subtraction and logic operations. In the last column, we computed the difference between estimated values and actual design values.

Table IV shows the result of the datapath estimation at the RT level for four benchmarks widely used by authors in high level synthesis. Design 4 is the *HAL*[13] already used to describe our algorithm. Design 5 is a *Band-pass filter* chosen from [15]. Design 6 is *AR recursive digital filter* borrowed from [16]. Finally, Design 7 is *Fifth order wave digital elliptical filter* from [17] chosen as a benchmark for the 1988 High-Level Synthesis Workshop. The second column shows the number of components (ALUs, MUXes and REGs) in datapaths of these examples.

Due to the unsufficient information about the layout cost reported in the literature, we are not able to compare our results against other datapath estimators or the actual designs. Note also that we assume the *word width* of these examples is 4, meaning all components in the library are designed to handle four-bit data as their inputs or outputs. The library for these examples consists of 4-bit *ALUs* such as multiplier or adder, different *multiplexers* and *register*. We ran the estimator requesting $A_r = 1$. However, achieving the exact value for A_r is not usually possible due to the fact that the interconnect area overhead produced by the folding phase depends on the coordinates of breaking points which is not predictable prior to folding. The estimator reports 41%-79% routing+empty area for these datapaths. In all cases the relative error $(\frac{Estimated-Actual}{Actual})$ of estimated results are within 12% of the actual area produced by *OCTTOOLS*.

VI. CONCLUSION

A fast layout estimation algorithm has been presented to be invoked iteratively in the different levels of the design process. To achieve this, we used non-probabilistic analytical formulas in a constructive algorithm. Our estimator can be used as an interaction tool between high and low level of design process to explore the search space in a guided fashion by using its prediction. The estimator quickly predicts the effect of our decisions in the layout level and helps the designer to get the minimum area.

† λ is a VLSI manufacturing technology scale factor. For example, in the Berkeley CAD tool (*OCTTOOLS*) the cell libraries can be scaled from 3.0 to 1.2 micron CMOS technologies.

ACKNOWLEDGMENT

The authors would like to thank Chad Winemiller and Phiroze Parakh who used OCTTOOLS to provide the layouts used for benchmarking our model.

REFERENCES

[1] A. Wu, V. Chaiyakul and D. Gajski "Layout-Area Models for High-Level Synthesis," in *Proc. ICCAD Conf.*, 1991.

[2] J. Weng and A. Parker "3D Scheduling: High-Level Synthesis with Floorplanning," in *Proc. 28th DAC Conf.*, 1991.

[3] F. Kurdahi and A. Parker, "Techniques for Area Estimation of VLSI Layouts," in *IEEE Trans. on CAD*, Jan. 1989.

[4] K. Ueda, H. Kitazawa and I. Harada, "Chip: Chip Floor Plan for Hierarchical VLSI Layout Design," in *IEEE Trans. on CAD*, Jan. 1985.

[5] X. Chen and M. Bushnell, "A Module Area Estimator for VLSI Layout," in *Proc. 25th DAC Conf.*, 1988.

[6] A. Dunlop and B. Kernighan, "A Procedure for Placement of Standard Cell VLSI Circuits," in *IEEE Trans. on CAD*, Jan. 1985.

[7] W. Dai and E. Kuh, "Simultaneous Floor Planning and Global Routing for Hierarchical Building-Block Layout," in *IEEE Trans. on CAD*, Sept. 1987.

[8] B. Preas and P. Karger, "Automatic Placement, a Review of Current Techniques," in *Proc. 23th DAC Conf.*, 1986.

[9] A. Wu and D. Gajski, "Partitioning Algorithms for Layout Synthesis from Register-Transfer Netlist," in *Proc. ICCAD Conf.*, 1990.

[10] G. Zimmermann, " A new area and shape function estimation technique for VLSI Layouts," in *Proc. 25th DAC Conf.*, 1988.

[11] M. Feuer, "Connectivity of random logic," in *IEEE Trans. Comput.*, Jan. 1982.

[12] F. Kurdahi and C. Ramachandran, "LAST: A Layout Area and Shape Function Estimator for High Level Synthesis," in *Proc. EDAC Conf.*, 1991.

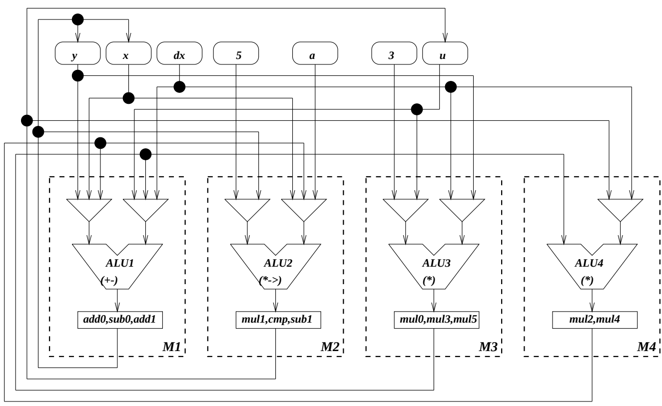
[13] P. Paulin and J. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's,"*IEEE Trans. on CAD*, June 1989.

[14] A. Casotto et al., "OCTTOOLS-5.1 User Guide," Electronics Research Laboratory, University of California, Berkeley, 1991.

[15] B. Green and L. Turner, "New Limit Cycle Bounds for Digital Filters," *IEEE Trans. Circuits and Systems*, April 1988.

[16] K. Parsad and C. Eswaran, "Limit Cycle Free Complex Biquad Recursive Digital Filters," *IEEE Trans. Circuits and Systems*, Feb. 1989.

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



(a)

Time Step	M1			M2			M3			M4		
	I11	I12	O11	I21	I22	O21	I31	I32	O31	I41	I42	O41
T=1				5	x	mul1	u	dx	mul0			
T=2	x	dx	add0				3	y	mul3	mul0	mul1	mul2
T=3	mul2	u	sub0	add0	a	cmp	u	dx	mul5	mul3	dx	mul4
T=4	y	mul5	add1	sub0	mul4	sub1						

(b)

Fig. 1. Datapath and interconnect information of HAL example

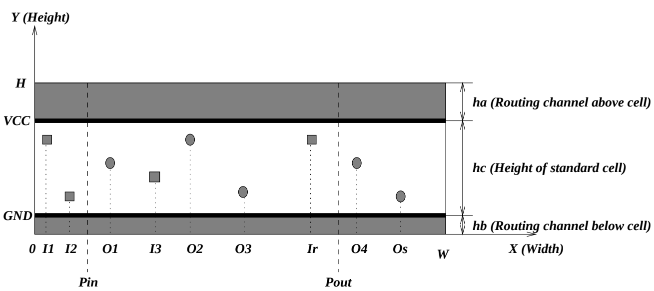


Fig. 2. Layout model for cells in the library

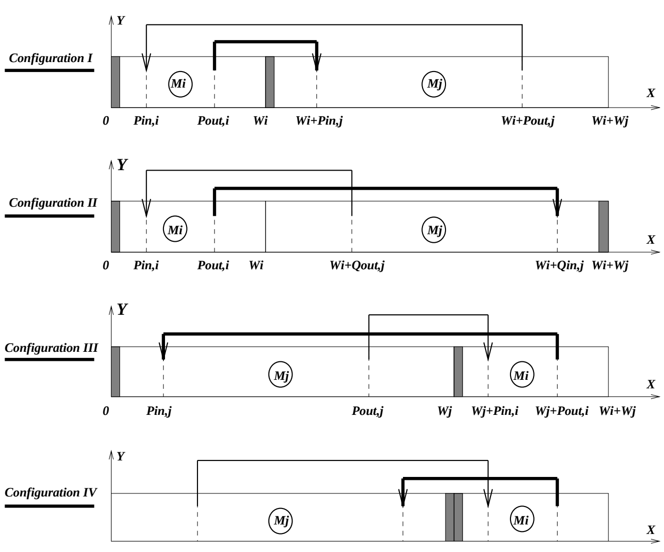


Fig. 3. Four configurations for two adjacent modules

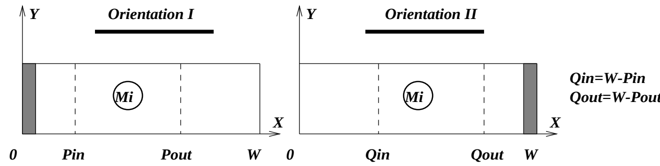


Fig. 4. Two orientations for each module

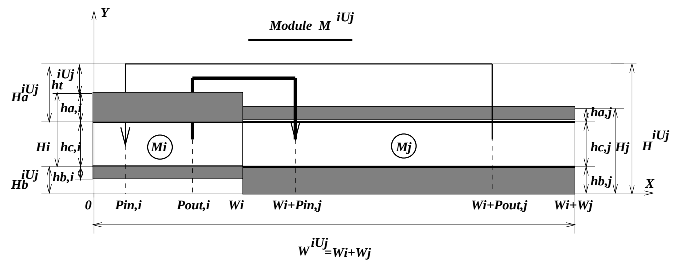


Fig. 5. Composed module ($M^{i\cup j}$)

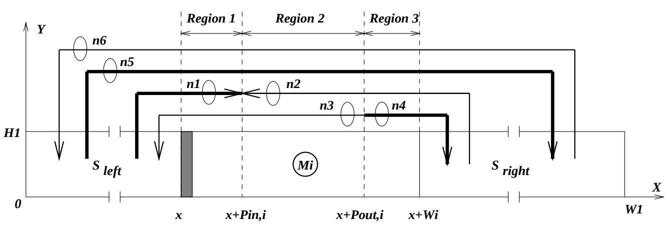


Fig. 6. Three regions above module (M^i) in one-row design

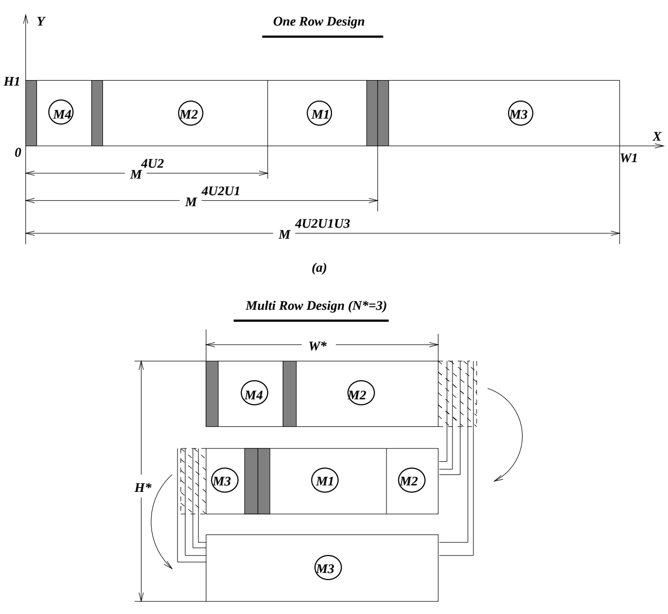


Fig. 7. One row design before and after folding