A Constructive Method for Data Path Area Estimation During High-Level VLSI Synthesis*

V. Natesan, Anurag Gupta, Srinivas Katkoori, Dinesh Bhatia, Ranga Vemuri

ECECS Department,P.O. Box 210030
University of Cincinnati
Cincinnati, OH 45221-0030, USA

Abstract— In this paper we present a fast and computationally efficient deterministic method for estimating the area of a Register Transfer Level datapath obtained during high level VLSI synthesis. The estimation makes use of a RT level netlist along with a pre-synthesized library of RT level components. The layout area is estimated using a quadratic programming based framework to get a quick module allocation and generating a topological floorplan which is then followed by heuristic algorithms for mapping RTL modules and their interconnections on a standard cell based layout design style. Experiments on a suite of benchmark examples show promising results with reliable accuracy.

I. Introduction

High level synthesis is the process of generating efficient register-transfer level (RTL) designs from algorithmic behavioral specifications. A typical high-level synthesis process can be broadly divided into two phases: data path synthesis and controller synthesis [2, 4]. Data path synthesis involves the generation of ALUs (arithmetic and logic units), storage devices (registers, register files, latches) and the interconnect structure (multiplexers, buses, and wires) that together comprise the data path. Data path components are usually selected from a parameterized and performance-characterized RTL module library. Control synthesis, the process of generating a finite state machine to control register transfer sequencing in the data path, usually follows data path synthesis. In order to select among alternative designs, the high-level synthesis algorithms make use of techniques to estimate area and speed (and possibly other attributes such as power dissipation). Both *stochastic methods*[8, 1] and a few *constructive methods*[22, 15] for estimating the layout area from RTL structures have been proposed. Accurate data path area estimation, usually based only on a partially formed data path structure, is crucial to the success of high-level synthesis. Many researchers have recognized the futility of simple minded estimates that add up individual RTL module areas and the importance of basing the area estimation on floor-planning and other physical design considerations [9, 7, 11]. Unfortunately, complete physical design can be quite expensive in computer time and is impractical to carry out during data path synthesis when many candidate designs have to be quickly evaluated. Hence, fast and accurate techniques for layout area estimation based on rapid generation of a floor-plan sketch that takes into account the effects of placement and routing are needed.

A good survey on the various techniques applied in interconnection analysis appears in [16]. Wireability analysis mainly deals with the problem of modeling the behavior of wires on a chip layout with the aim of predicting the amount of space to be allocated for wiring the interconnections. Almost all the wiring models proposed so far are stochastic in nature. Probabilistic estimation techniques make some assumptions on the interconnection length distributions and predict the layout area without taking into account the physical details of each design[18, 8]. Some constructive methods recursively partition the input design upto the leaf level where the shape and size of the cells are known. The cells are then combined up to the root level to generate the complete chip. Zimmerman[22] uses such a constructive technique. He describes a model for the prediction of shape functions of the modules in a hierarchical manner. The model assumes a slicing geometry and also assumes that a structural hierarchy exists between the modules. The problem with these constructive methods is that they are slow. A layout estimation algorithm for RTL datapaths has been proposed by Nourani and Papachristou[15] which predicts the layout model using analytical formulas in a constructive algorithm. This algorithm first places the modules in a single row and then folds the row to obtain the desired aspect ratio of the chip. The authors report an average error of about 10-12% on their experiments. Mecha et al.[12] provide a method to estimate the layout area of datapaths during the high level synthesis process. They develop different heuristics to estimate close and distant interconnects and drive the design process towards better designs in an iterative manner. They report an average error of 5% on small-scale examples. They also provide a brief survey on the various area estimation approaches.

Many of the techniques proposed so far are time consuming in execution. Also, the accuracy of stochastic estimates is hard to establish. Motivated with the goal of

*This research is partially supported by the solid state electronics directorate at Wright Laboratories of the United States Air Force under contract number F33615-91-C-1811.

ASP-DAC '97
0-89791-851-7$5.00

layout area estimation, we propose a fast, deterministic and, constructive method for estimating the layout area of data paths produced during the high level synthesis. Our method is very useful for quick area estimation of RTL datapaths. Also, the proposed method can be utilized for quick elimination of designs that may have excessively large area, thus reducing the design space. Our approach makes use of a new quadratic-programming based technique for generating an initial floor-plan, an iterative readjustment technique based on topological constraint reduction to eliminate module overlaps in the floor plan, a module reshaping heuristic to squeeze the floor-plan, and accurate track and feed-through estimates to determine the layout area. We present experimental results on several examples which show that our area estimates deviate from actual areas by 5% on average.

This paper is organized as follows. Section II gives an overview of the proposed estimation technique. Section III details the characterization of the RTL library components. The quadratic programming based module allocation is detailed in section IV. The floorplanning technique appears in section V followed by the area estimation heuristic in section VI. The experimental results and conclusions appear in sections VII and VIII respectively.

II. Overview

The proposed area estimation technique takes a net-list of register-transfer level modules whose individual bounding box areas are known (but not necessarily their aspect ratios) and estimates the overall bounding box area of the net-list based on the generation and evaluation of a floor-plan sketch which takes into account the overheads of placement and routing. Our technique has the following significant merits.

- A novel mathematical programming and constraint-driven floorplanning based technique is used
- The design-style specific area estimation heuristic takes into account the effects of placement and routing
- The framework is constructive and uses an extremely fast approach
- Efficient in accurately estimating the actual layout area

Briefly, our method can be described as follows. The behavioral level specs of a design are provided as input to Distributed Synthesis System(DSS)[17]. DSS produces different RT Level descriptions for the design. The RTL library cells are characterized as explained in section III. The RTL design file and the library information are input to a quadratic programming(QP) based module allocation procedure (section IV). This results in a topological distribution where the modules are spread over the layout area. Due to an assumption made during QP that modules are point sized objects, the module distribution will have a lot of overlapping between modules and thus this is not a feasible solution. Therefore, the output of the QP based placement is considered to specify only the relative topological placement of the modules. The topological ordering of the modules is used as input to a constraint-based floorplanning method (section V). The floorplanning step results in an overlap-free distribution of the modules. This distribution forms an input to a layout area estimation procedure (section VI). The layout area can be estimated for each RT level description of a given design and the best schedule can be identified and taken to the layout level.

Thus, our approach can be viewed upon as a *filter* that identifies the area-wise good designs from a set of different RTL schedules for a given design. Typically, a designer is left with a vast design-space to explore and such a filter will help the designer make valuable design decisions, thus saving considerable time.

III. Area Characterization of the RTL Module Library

The module library contains parameterized register-level modules such as n-bit registers, n-bit adders and n-bit m-to-1 multiplexors. Modules are parameterized with respect to number of inputs where applicable and bit-width of each input. The library contains interface descriptions of each module, description of its parameters and area characteristics.

Area of a module instance is the bounding box area of its layout implementation. The area characteristic of a given library module is a function of its area with respect to the module's parameters such as bit width and input-size. Each library module is characterized for area by actually generating layouts for several instances of the module with different parameter values. It should be noted that the area of RTL modules is comprised of standard cell area and intra module wiring. For our estimation purpose, we make use of several such modules and their interconnection as defined by the RTL netlist.

Although, in this work, the area values are computed for 2μ CMOS technology, they can be linearly scaled for other feature sizes since the standard cell library is a scalable CMOS library. Lager IV Silicon Compiler [20] is employed in synthesizing the layouts from the RTL descriptions.

TABLE I
Area Characteristics for Parameterized Library Modules (BitWidth $\geq$ 1)

Module	(BitWidth-Layout Area (x$10^3 sq.\ micron$))
Not	1-4.59, 2-9.21, 4-15.36, 8-28.99, 16-65.65
Or	1-7.20, 2-16.12, 4-32.03, 8-56.37, 16-134.50
And	1-6.43, 2-14.33, 4-25.92, 8-58.23, 16-117.88
Nand	1-5.66, 2-12.54, 4-22.88, 8-42.75, 16-100.46
Nor	1-5.66, 2-12.54, 4-21.60, 8-42.75, 16-105.60
Xnor	1-7.96, 2-17.92, 4-36.60, 8-72.52, 16-156.28
Xor	1-7.96, 2-17.92, 4-36.60, 8-70.28, 16-147.01
Adder	1-23.29, 2-47.48, 4-90.36, 8-177.00, 16-388.51
Subtractor	1-26.62, 2-48.96, 4-109.87, 8-182.40, 16-398.24
Comparator	1-25.08, 2-54.75, 4-102.76, 8-248.64, 16-582.62
Multiplier	1-26.62, 2-107.52, 4-388.48, 8-1668.74, 16-8713.70
Divider	1-28.28, 2-107.52, 4-434.52, 8-1764.84, 16-9314.28
ShiftRegister	1-74.78, 2-123.90, 4-204.25, 8-409.53, 16-831.09
Latch	1-28.28, 2-50.59, 4-120.33, 8-222.04, 16-402.52
Signal	1-157.44, 2-251.32, 4-343.26, 8-543.60, 16-1089.29
Multiplexor	2-to-1: 1-11.52, 2-23.04, 4-45.79, 8-76.89, 16-1180.08
	4-to-1: 1-25.87, 2-56.65, 4-114.00, 8-205.84, 16-471.20
	8-to-1: 1-66.31, 2-126.00, 4-225.93, 8-481.46, 16-1149.79

Table I shows the area characteristics for some of the library modules. For each module, its area characteristic is presented in a tabular form where the first entry is the bit width and the second entry is the area of the module instance measured in square micron. Interpolation/Extrapolation is assumed for the parameter values that are missing in the characterization table.

IV. Quadratic Programming based Module Allocation

The objective here is to obtain a relative ordering of the modules based on their connectivity. The interconnections between the RTL modules are represented in the form of a quadratic function, which is solved to obtain the positional coordinates of the modules. The QP based method for detailed standard cell placement has been used widely[13, 19, 14]. The Quadratic Programming(QP) based technique has two distinct advantages: 1) it is possible to find a solution close to the global optimal solution and 2) the solution can be obtained in a very short time.

Let $\mathcal{M} = \{M_1, M_2, \ldots, M_n\}$ be the set of n RTL modules. Let the I/O modules be represented by the set $IO = \{IO_1, IO_2, \ldots, IO_m\}$. The positions of the I/O modules are assumed to be fixed. Let the position of the center of module $i \in \{\mathcal{M} \cup IO\}$ be represented by a tuple (x_i, y_i) in the XY-plane. For the sake of simplicity all the pins of a module are approximated to the center of the module. Also, let $\mathcal{N} = \{N_1, N_2, \ldots, N_k\}$ be the set of k nets in the design. The circuit is viewed as a hypergraph and thus each net defines interconnections among RTL modules and I/O modules. Typically in a RTL netlist, the net is parameterized for a bit-width. In our implementation we represent bit-width by that many number of connections between the RTL modules connected by the net. For each net, there appears term(s) in the objective function representing its connectivity. Thus, if modules C_i and C_j are connected by the net N_t, $1 \leq t \leq k$, then the term $\left\{(x_i - x_j)^2 + (y_i - y_j)^2\right\}$ is introduced in the objective function. Thus, the objective function can be written as,

$$\sum_{t=1}^{k} \sum_{M_i, M_j \in N_t, i \neq j} \left\{(x_i - x_j)^2 + (y_i - y_j)^2\right\}.$$

A multi-terminal net of size p (i.e., a net connecting to p modules) introduces $p(p-1)/2$ terms in the objective function. The minimization of objective function forces a minimization of placement of modules that form a clique of size p. Thus multi-terminal nets with large size will introduce a bias in the objective function. Therefore, a suitable weighting has to be introduced to eliminate unnecessary biasing in the objective function. For each net N_t, a weighting factor $f_t = 2/p$, where p is the number of modules in net N_t, is introduced. Let $a_1, a_2, \ldots, a_n$ be the area of n modules and let (x_c, y_c) be the coordinates of the center of the layout. Then, the constraints,

$$\textstyle\sum_{i=1}^{n} a_i x_i = A x_c, \sum_{i=1}^{n} a_i y_i = A y_c,$$

are introduced. These constraints force a uniform distribution of modules. Here $A = \sum_{i=1}^{n} a_i$. The quadratic programming based placement with modified objective function can be stated as,

$$\textbf{Min} \sum_{t=1}^{k} f_t \left(\sum_{M_i, M_j \in N_t, i \neq j} \left\{(x_i - x_j)^2 + (y_i - y_j)^2\right\} \right)$$

$$\textbf{subject to} \sum_{i=1}^{n} a_i x_i = A x_c, \sum_{i=1}^{n} a_i y_i = A y_c.$$

The above formulation is a standard quadratic programming problem with linear equality constraints. There are several methods that can be used to solve this QP problem[10]. We use the *conjugate gradient* method[5] which produces solution to the QP problem in a very short time.

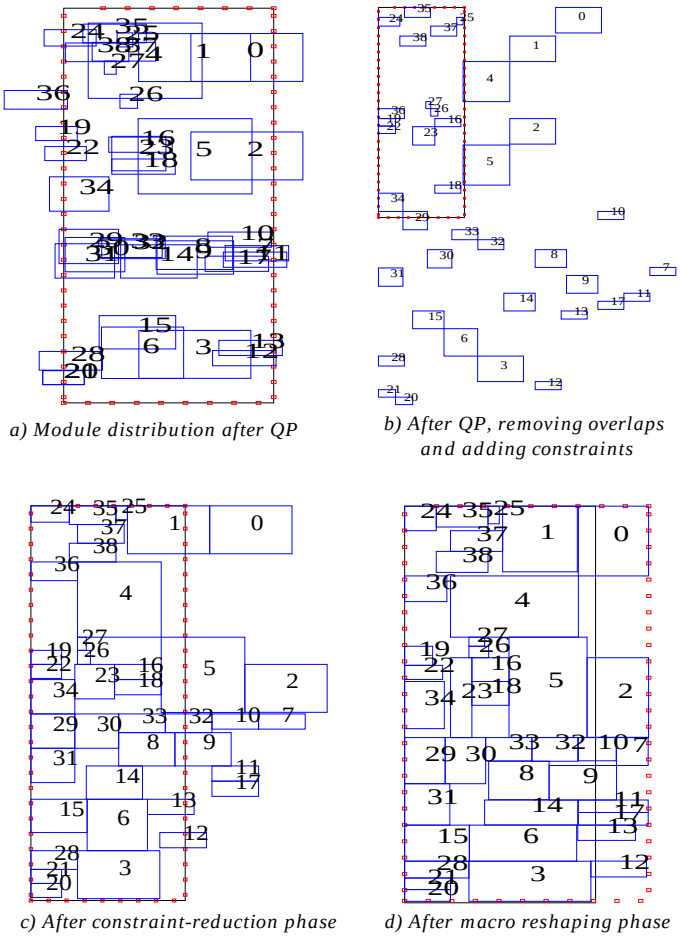
The solution to the above formulation seems trivial since the cost function will equal zero if $x_i = x_j, 1 \leq i, j \leq n$ and $y_i = y_j, 1 \leq i, j \leq n$. However, some x_i and y_i represent the co-ordinates of fixed IO modules around the periphery of the layout. These IO modules exert an outward pull on the modules connected to them and thus ensure that the solution is not a trivial one.

The QP based minimization views each module as a point sized object and thus the resulting solution will result in a module distribution with mutual module overlaps. This is true since each module has an area associated with it. In our experience we have found that the solution to the QP formulation results in a module distribution where modules are distributed around the center of the layout but do not have uniform distribution over the entire layout area. The equality constraint favors distribution around the center, while the minimization creates an inward pull for all of the modules.

In order to minimize overlap and spread the modules over the entire layout area, the module set is bipartitioned using a simple partitioning scheme. The bipartitioning is done with respect to the module distribution obtained after solving the quadratic programming based formulation. Thus module allocation to a partition is favored by its geometrical co-ordinate position as produced by the solution of the quadratic program. It is ensured during bipartitioning that the area of the two partitions are about the same. The modules are then biased towards the center of the newly partitioned regions. The global position of modules (obtained at the end of quadratic optimization) determines the bias of a module towards the center of the corresponding partition. At the end of the partitioning phase, the constraint set is reformulated. The center-spread constraints of the modules are formulated for each region R_i, $i = 1, 2$. The new center-spread constraints can be stated as,

$$\textstyle\sum_{q=1}^{2} \sum_{i \in R_q} a_{i,q} x_i = A x_{c,q}$$

where $a_{i,q}$ represents the area of module i in region R_q



a) Module distribution after QP

b) After QP, removing overlaps and adding constraints

c) After constraint-reduction phase

d) After macro reshaping phase

Fig. 1. Illustrating the various stages of the floorplanner

and $x_{c,q}$ represents the x co-ordinate of the center of region R_q. Similar constraint for the y co-ordinate is also applied. The problem with the new center-spread constraints is again solved for the global optimal positions for the modules. This will result in positions for the modules such that the constraints biasing the two groups of modules to their regions is satisfied. Since the center of the two regions are away from each other, this solution will be more spread out than the first level of optimization which had just one centering constraint.

This process is carried to multiple levels of optimization. After each level of global optimization, the number of partitions(regions) is doubled and for each partition, the centering constraints are formed for the cells that belong to it. Global optimization followed by partitioning is iterated until only a few modules are left in each region. The solution also spreads out across the layout area due to the bias of each cell towards its respective center. At this point, the modules are fairly spread on the area of the layout. The mutual overlaps for various modules still have to be resolved. Refer figure 1.a which shows the output after the solution to QP is obtained for an example design. This is not a feasible floorplan due to the overlaps between the modules. In the following section we discuss the way to achieve a non-overlapping floorplan for the design.

V. Constraint based FloorPlanning

The QP based placement is only a relative placement step. The relative placement is used to form a set of topological constraints with respect to the positioning of the RTL modules in the two-dimensional plane. To obtain an overlap-free floorplan, a modified constraint-based floorplanning method proposed by Vijayan and Tsay[21] is used.

If the QP based placement has resulted in a placement where module M_i is placed to the left of module M_j, then a horizontal constraint is added to the constraint set. Vertical constraints are formulated in a similar way for top to below relationships. If there is a vertical constraint between M_i and M_j and if these two modules are overlapping, then M_j can be placed just below M_i, abutting M_i so that there is no overlap between M_i and M_j *and* the constraint is satisfied. A relaxed, non-overlapping floorplan of the modules can be formed by repeating this process for each constraint in the constraint set. Figure 1.b shows an example where the overlaps have been removed.

If a pair of modules M_i,M_j have constraints in both the vertical and horizontal directions, then the constraint set is said to have redundant constraints between M_i and M_j. The basic idea with the approach in [21] is to remove redundant constraints. The removal of a constraint can amount to a reduction in the floorplan area. After removing the redundant constraints, the floorplanner reshapes the modules so that the area can be further minimized. The constraint reduction is explained in section A and section B deals with module reshaping.

A. Constraint Reduction

The goal here is to remove some redundant constraints so that the floorplan area is minimized. A horizontal constraint graph G_H and a vertical constraint graph G_V are formed from the initial relative placement of the modules. G_H and G_V are directed acyclic graphs. The *length* of a path of blocks in G_H or G_V is defined to be the sum of the dimensions of the blocks and the separations(if any) between them. The path with the longest length is termed the *critical* path.

The graphs G_H and G_V can be topologically sorted based on the module positions and the critical path can be constructed by scanning the sort and assigning the smallest possible value for each module subject to the constraints between the modules. The constraint reduction phase repeatedly chooses the most critical path and removes a strongly redundant edge on the selected critical path. By removing a redundant edge, the critical path is broken into two smaller paths, thus yielding a compacted floorplan. The redundant edge removal is carried on until no more redundant edges remain. The result of the constraint removal phase will be a floorplan whose area is minimized. An example output at the end of the constraint reduction phase is shown in figure 1.c.

B. RTL Macro Reshaping

The reshaping heuristic also selects the critical paths in an iterative fashion. The dimensions of each module on the selected critical path are reduced by a user-specified dimension reduction factor. It is ensured that the area of the modules remain a constant during the reshape process. After the reshaping of the modules, the area of the floorplan is calculated by traversing G_H and G_V and the module shapes are updated if the new floorplan resulted in a smaller area. Repeating the reshaping process for a user-specified number of iterations results in a compact floorplan for the design. Figure 1.d illustrates the floorplan after the reshaping phase. Please note that the position of the I/O modules have been adjusted in figure 1.d to account for the new area of the floorplan.

A single run of the constraint reduction and the reshaping phases constitutes one pass of the floorplanning heuristic. For a given design, typically 3 or 4 passes are made which results in a very compact floorplan.

VI. Area Estimation

The floorplanning technique that has been described so far will help in estimating the layout area of the modules only, ie., it does not include the estimate for the routed interconnect wires between the modules. A proper layout estimator should also be able to predict the routing requirements. The routing area estimation is explained in this section.

The DSS tool[17], after synthesizing the RTL design for the input, invokes the Design Manager (DMoct) of the Lager[20] tools. The lager tools flatten the RTL blocks and come up with a standard-cell layout for the given design. Our area estimator is developed to predict the area of the standard-cell layout. Since we are going to predict the area with respect to a standard-cell layout as a target layout style, we are justified in the reshaping of the pre-synthesized RTL blocks. The fact that we target a standard-cell layout style forms the basis for the proposed wire area estimation heuristic.

A. Problem Formulation

Assume that there is an underlying set of standard-cell rows and interlacing channels over the entire layout area. The height of the rows are taken to be equal to the height of the leaf cells in the library. The channel height is either user defined or by default they can be taken to be equal to that of the standard-cell rows. All the channels are assumed to be of the same height to start with. The correct height of each channel can be obtained after executing the estimating heuristic. Each channel consists of a number of segmented tracks. Figure 2 illustrates the above concepts.

Let $\mathcal{C} = \{C_1, C_2, \ldots, C_q\}$ be the set of q channels. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_w\}$ be the set of w rows. For each channel $c \in \mathcal{C}$, let $\mathcal{T}_c = \{T_c^1, T_c^2, \ldots, T_c^z\}$ be the set of z_c tracks in channel c. Let $\mathcal{S}_{T_c} = \{S_{T_c}^1, S_{T_c}^2, \ldots, S_{T_c}^x\}$ be the set of T_{c_x} segments of track T_c in channel c. The goal here is to obtain the layout area by estimating the number of tracks z_c required for each channel c, $1 \leq c \leq q$ and by estimating the number of feedthroughs f_r required for each row r, $1 \leq r \leq w$. The number of tracks in each channel can be estimated by first estimating the number of segments required for each net. The estimation heuristic is explained in the following section.

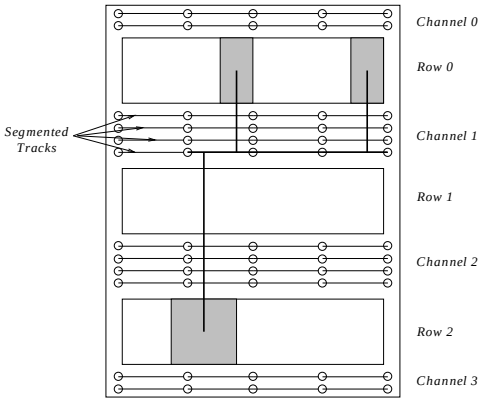


Fig. 2. Layout model for wire area estimation

B. Track Estimation

For each net $d \in \mathcal{N}$, the mean value $\overline{Y_d}$ in the vertical direction is computed, based on the Y co-ordinate positions of the modules connected to d. Thus, if $M_d^1, M_d^2, \ldots, M_d^k$ is the set of k modules connected to net d, $\overline{Y_d}$ is given by,

$$\overline{Y_d} = \frac{M_d^1 + M_d^2 + \ldots + M_d^k}{k}.$$

The channel c closest to $\overline{Y_d}$ is identified. Let X_d^{left} and x_d^{right} be the left-most and right-most co-ordinate points of the modules connected to net n. From X_d^{left} and X_d^{right}, the segments $\{S_{T_c}^i, S_{T_c}^{i+1}, \ldots, S_{T_c}^j\}$ occupied by net d in some track T_c^k, $1 \leq k \leq y_c$ can be computed. Let $\mathcal{S}_d$ be the set of segments occupied by net d. This procedure is repeated for all the nets in the design. The number of tracks required in each channel is then computed from $\mathcal{S}_d$, $1 \leq d \leq n$, using the left-edge algorithm proposed by Hashimoto and Stevens[6].

The left-edge algorithm used was a simple one without using doglegs. This resulted in quite a few under-utilized tracks, that is there were tracks which had only few of their segments used by some net or the other. This resulted in a lot of wasted space on these under-utilized tracks. In practical layouts, under-utilization is countered by introducing doglegs and by local improvements. The effect of under-utilized tracks was significant for area estimation of large designs. In order to obtain better and closer estimates, we pruned the routing space using a simple heuristic. Thus, after the assignment of all nets to tracks, if track T_c in channel c consists of s segments of which only $s1$ segments are occupied by some nets and if $s1/s < k$ (k is a constant in the range of 0 to 1), then we

ignore the track T_c. The number of tracks z_c in channel c is then reduced by 1. Experimentally we found k to be $\simeq 0.25$.

C. Feedthrough Estimation

To estimate the Feedthrough requirement for each row, the following heuristic is adopted. Let $\mathcal{M}_d = \{M_d^1, M_d^2, \ldots, M_d^m\}$ be the set of m_d modules connected to net d and let $\{Y_d^1, Y_d^2, \ldots, Y_d^m\}$ be the corresponding Y co-ordinate position of the modules in $\mathcal{M}_d$. For each module $k \in \mathcal{M}_d$, a vertical Steiner link L_k is drawn between Y_d^k and $\overline{Y_d}$. The row set $\mathcal{R}_{L_k} = \{R_{L_k}^i, R_{L_k}^{i+1}, \ldots, R_{L_k}^j\}$ through which the link L_k passes can then be computed. The number of feedthroughs f_r required for each row $r \in \mathcal{R}_{L_k}$ is incremented by one. The feedthrough estimation procedure is repeated for all the nets in the design.

D. Layout Area Estimation

Figure 2 illustrates a net which connects two modules in the Row 0 and a module in Row 2. The horizontal trunk of the Single-trunk Steiner tree for the net passes through Channel 1 and it can be seen to occupy three segments of a track in Channel 1. Also, since the vertical Steiner link connecting module 1 to the horizontal trunk passes through Row 1, the feedthrough count for Row 1 is incremented by one. A similar procedure is applied for all the nets in the design.

The trackwidth W_t and the feedthrough width W_f are assumed to be specified in a technology file. The height H_c of each channel c, $1 \leq q$ is then calculated as $z_c \times W_t$ where z_c is the number of tracks in channel c. (Please note here that we have assumed that track width also includes the separation distance between tracks). The total chip height H can be computed as,

$$H = \sum_{c=1}^{q} H_c + \sum_{r=1}^{w} H_r,$$

where H_r is the height of row r. The width W of the layout is computed as,

$$W = W_{FP} + max(f_r \times W_f), 1 \leq r \leq w.$$

Here W_{FP} is the width of the layout obtained at the end of the floorplanning step. The layout area A can then be estimated as the product of H and W.

VII. Results and Analysis

The proposed area estimation technique has been implemented as a stand-alone software module and, for experimentation purposes, interfaced with the data path generation module of the DSS high-level synthesis system [17]. Our area estimation module is used at various steps during the data path synthesis process in DSS: during scheduling, during register optimization and during interconnect optimization. In each case, bounding box area of the partially formed data path is estimated for each alternative data path structure being explored.

To evaluate the effectiveness of the proposed estimates, we have compared our area estimates with the actual area after the data path is placed and routed using the standard-cell placement and routing tools in the Lager IV [3] layout synthesis system. For fair comparison, both the actual layout generation and the estimation is done for fully formed data paths following data path synthesis in DSS for each of the examples shown in Table II. Traffic Light Controller (TLC) is used for regulating traffic at a road intersection. Compress and Decompress implement a simple look-up table based compression and decompression algorithms respectively. Find implements a sort and search algorithm in hardware. FIFO is a first-in first-out queue. Diffeq is a differential equation solver. Elliptic is a fifth order elliptic wave filter. Viper is a simple 16 bit microprocessor. The behavioral descriptions of the above benchmark suite employ various high level constructs such as conditionals (TLC, FIFO), loops (Find,FIFO), subprograms (Compress, Decompress, Viper), besides straight line code (Elliptic, Diffeq).

The results are presented in Table II. The column RTL Area in these tables refers to the sum total of the area of RTL modules. This area includes the area of the standard cells that belong to each RTL modules as well as the area of the intra-module interconnections. A1 in the table refers to the estimated area obtained after executing our heuristic and A2 refers to the actual layout area. All the areas are in square microns. The last two columns in Table II shows the percentage of error between the estimated area and the actual layout area and the run-time for layout area estimation in seconds.

In all cases, our estimates differ from the actual area by an average of about 5%. Given that some of our designs are very large, an error of 5% is quite promising. What is more important to note is the time spent in finding the estimated area. For example, the layout area estimation for `viper` took a little over a minute of cpu time on a SUN Sparc-5 against well over five hours to compute actual area on a SUN Sparc-20. This is significant since this estimation technique can be used to evaluate multiple RTL schedules for a given design and filter the bad quality schedules without going through the time consuming task of layout generation. The estimates cannot be readily compared with other works due to the difference in synthesized designs and lack of proper frameworks. However, some of the examples that we have used are the *largest* and our area estimates are close to the actual area.

VIII. Concluding Remarks

A fast and deterministic layout area estimation has been proposed and implemented for use during the high level VLSI synthesis. The execution time for estimating the layout area is very small and thus the tool can be used to estimate areas for various design alternatives at various stages of the high level synthesis process. Over a range of examples, the error between the estimated area

TABLE II
RESULTS OF ESTIMATION

Design	Blocks	I/O	Nets	StdCells	Transistors	RTL Area $\times 10^6$	Estimated Area (A1) $\times 10^6$	Lager Area (A2) $\times 10^6$	$\frac{\|A1-A2\|}{A2}$ %	run time *sec*
TLC	33	56	110	306	2694	2.207	2.448	2.573	4.86	4.9
decompress	35	54	175	433	4116	2.972	4.605	4.405	4.54	6.2
compress	37	55	197	479	4345	3.267	5.481	4.883	12.25	6.4
find	60	97	299	893	9936	7.858	12.525	13.460	6.95	19.3
diffeq	66	126	381	1009	12194	10.789	15.600	16.401	4.88	39.5
fifo	51	139	576	1289	17524	20.380	31.103	32.203	3.42	23.7
elliptic	100	215	598	1688	20564	16.691	35.211	34.158	3.08	118.0
viper	81	158	816	2509	30442	25.189	63.437	62.412	1.65	64.5

and the actual layout area is very small. Except for two out of eight designs, the error was contained within 5% and the average error is also no more than 5%. We believe that such early and quick estimates can help designers in speeding up the design decisions.

It should be noted that the technique is generic and need not be applied only to standard cell based layout styles. The QP formulation and the constraint-driven floorplanning remain the same irrespective of the underlying layout style. For example, data paths are often organized into slices to simplify routing overhead. The area estimation heuristic explained in section VI can be changed suitably to reflect the placement and routing in such cases.

Currently, the input netlist file that we have gives the connectivity information on each net, even if that net belongs to a bus. If it is required that the modules connected to a bus should remain together, i.e., if the bit-sizes of the nets are to be considered, it can easily be incorporated into the model by good net-weighting schemes. Also, module level placement requires buffering for long signal lines. The buffers can be modeled into the QP formulation by collapsing the buffers into the modules that drive them so that the placement of modules and their buffers will always be together.

The current implementation deals with area estimation of RTL datapaths and acts as a filter in choosing good RTL schedules from a large set of schedules based on the area. As part of our on-going work we are investigating into developing filters which would consider performance driven parameters such as delay and power by computing accurate estimates of delay and power. This would help the designer to base the decisions based on multiple design objectives.

REFERENCES

[1] Xinghao Chen and Michael L. Bushnell. A module area estimator for vlsi layout. In *Proceedings of the 25th Design Automation Conference*, pages 54–59, 1988.

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

[3] R. W. Broderson ed. *Anatomy of a Silicon Compiler*. Kluwer Academic Publishers, 1992.

[4] D.E.Thomas et al. *"Algorithmic and Register Transfer Level Synthesis: The System Architect's Workbench"*. Kluwer Academic Publishers, 1990.

[5] G.H. Golub and C.F. Van Loan. *Matrix Computations*. Baltimore MD: Johns Hopkins University, 1985.

[6] A. Hashimoto and J. Stevens. Wire routing by optimizing channel assignment with large apertures. In *Proceedings of 8th Design Automation Workshop*, pages 155–169, 1971.

[7] Rajiv Jain. High-level area-delay prediction with application to behavioral synthesis. Technical Report CENG 89-23, University of Southern California, 1989.

[8] F. J. Kurdahi and A. C. Parker. Technique for area estimation of vlsi layouts. *IEEE Transactions on Computer-Aided Design*, 8(1):81–92, January 1989.

[9] Fadi Joseph Kurdahi. Area estimation of vlsi circuits. Technical Report CRI-87-40, University of Southern California, 1987.

[10] David G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, 1993.

[11] M.C. McFarland. Using Bottom-Up Design Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. In *Proceedings of 23rd Design Automation Conference*, 1987.

[12] Hortensia Mecha, Milagros Fernandez, Fransisco Tirado, Julio Septien, Daniel Mozos, and Katzalin Olcoz. A method for area estimation of data-path in high level synthesis. *IEEE Transactions on Computer-Aided Design*, 15(2):258–265, February 1996.

[13] Jurgen M.Kleinhans, George Sigl, Frank M.Johannes, and Kurt J.Antreich. GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization. *IEEE Transactions on Computer-Aided Design*, 10(3):356–365, March 1991.

[14] V. Natesan and Dinesh Bhatia. Performance driven placement for cell-based designs. In *IEEE International ASIC Conference and Exhibit*, pages 237–240, 1995.

[15] Mehrdad Nourani and Christos Papachristou. A Layout Estimation Algorithm for RTL Datapaths. In *Proceedings of 30th Design Automation Conference*, pages 285–291, 1993.

[16] Bryan Preas and Michael Lorenzetti, editors. *Physical Design Automation of VLSI Systems*, chapter 2. The Benjamin/Cummings Publishing Company, Inc., 1988.

[17] Jayanta Roy, Nand Kumar, Rajiv Dutta, and Ranga Vemuri. DSS: A Distributed High-Level Synthesis System. *IEEE Design and Test of Computers*, June 1992.

[18] Sarma Sastry and Alice C. Parker. Stochastic models for wireability anaysis of gate arrays. *IEEE Transactions on Computer-Aided Design*, 5(1):52–65, January 1986.

[19] Arvind Srinivasan, Kamal Chaudhary, and Ernest S.Kuh. RITUAL: A Performance-Driven Placement Algorithm. *IEEE Transactions on Circuits and Systems -II: Analog and Digital Signal Processing*, 39(11):825–839, November 1992.

[20] University Of California, Berkeley. *LagerIV Release 4.0*, 1991.

[21] Gopalakrishnan Vijayan and Ren-Song Tsay. A new method for floor planning using topological constraint reduction. *IEEE Transactions on Computer-Aided Design*, 10(12):1494–1501, Decemeber 1991.

[22] Gerhard Zimmerman. A New Area and Shape Function Estimation Technique for VLSI Layouts . In *Proceedings of 25th Design Automation Conference*, pages 60–65, 1988.