Analytical Approach to Custom Datapath Design

Serkan Askar *Maciej Ciesielski*
Department of Electrical & Computer Engineering
University of Massachusetts, Amherst, MA 01003
{*saskar, ciesiel*}*@ecs.umass.edu*

Abstract— **This paper addresses the problem of layout design automation of datapath cells. We present a novel approach to transistor placement problem for custom datapath designs and demonstrate that it can be applied to practical designs. Our approach is based on an analytical model which employs a mixed integer linear programming (MILP) technique. The novelty and originality of the method is the efficient management of the complexity of the underlying mathematical model. Our prototype tool automatically handles transistor merging, folding, and intra-cell component sharing.**

I. Introduction

Layout generation for high-performance datapaths has been typically done manually due to stringent requirements imposed on area and performance of such circuits. So far, datapath layout did not enjoy the benefits of design automation due to high correlation between quality of layout and circuit performance. However, as the complexity of datapath structures grows the requirement on turnaround time becomes critical, which makes datapath layout automation inevitable.

Datapaths are characterized by highly regular layout structures. Typical datapath floorplan consists of an array of bit slices and words of identical bit cells. Since each bit slice is replicated a number of times (determined by the datapath width) with very little or no modification, layout generation of such regular structures reduces to a careful design, often by means of hand-crafting, of individual cells.

This paper addresses the problem of automating layout design of datapath cells. Specifically, it concentrates on the transistor placement problem. Although placement and routing are tightly interrelated, they are often performed as separate design phases in order to manage enormous design complexity. Furthermore, modern multi-layer routing technology makes it possible to route layout structures with diffusion-limited placement, provided that sufficient spacing for contacts is provided to facilitate the subsequent routing. Due to high repetition of datapath cells the task of datapath layout optimization reduces to minimizing the area of individual cells, subject to routability and performance constraints.

Several techniques and algorithms have been proposed in the past to automate transistor placement. These algorithms can be broadly classified as *deterministic* and *stochastic*. Among the deterministic algorithms, force directed, eigenvalue, and geometry-based constructive methods have been used extensively [1], [2], [3], [4]. Stochastic methods are represented by simulated annealing and genetic algorithms, which are known for their ability to handle multi-dimensional cost function [5]. Place & route tools, such as KOAN/ANAGRAM [6] and PUPPY [7], designed specifically for analog devices, are best examples of layout techniques based on simulated annealing.

Our approach to custom datapath layout is based on a deterministic analytical model using mathematical programming techniques. We present a novel approach to transistor placement problem and demonstrate that it can be applied to practical datapath designs. Our method is based on an analytical approach which employs a mixed integer linear programming (MILP) technique. MILP has been previously proposed for floorplanning and cell placement [8] and is known to suffer from computational complexity problem. The novelty and originality of our method is the efficient management of the complexity of the underlying mathematical model.

This work was supported by a grant from Compaq's Alpha Development Group

II. Datapath Design Problem

A number of constraints is imposed on the design of datapaths, whether it is done manually or automatically. These constraints must properly capture the requirements imposed by the global datapath floorplan, component geometries, and technology requirements. This section provides a brief overview of the constraints and objectives in the automated design of custom datapath.

A. Bit-slice Constraints

Figure 1 shows an outline of a typical datapath cell in a vertical orientation. The width (*pitch*) of the bit-slice is fixed, typically delimited by VDD/VSS power supply rails. Signal nets are connected to the cell components by means of *bristles*. Vertical bristles are *data lines*, providing wiring between different functional cells within the same bit slice. Horizontal bristles provide *control lines* between functionally identical cells of different bit-slices. Data lines run in parallel to the power rails. Control lines span the width of the datapath in the direction perpendicular to the power rails. The physical location of the bristles is typically known prior to placement of components in the cell.

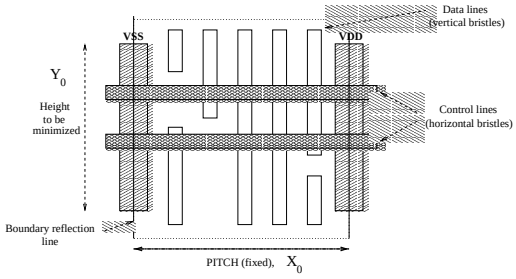

Fig. 1. Representation of a datapath cell

B. Component Geometry Constraints

Each component represents a single transistor or a chain of transistors to be placed as one entity (called *placeable component*). Physical constraints imposed on placeable components allow for certain degree of freedom, including multiple shapes and orientations.

Multiple instances and transistor folding: Each placeable component can take one of several predefined shapes (in our case limited to rectilinear polygons). These different shapes come from different chaining of transistors or from transistor folding, resulting in rectangles with different aspect ratios. The aspect ratios of the folded components are computed in our preprocessing program and considered in the MILP model as different instances of the component. The best instance of the component is automatically selected during the placement procedure.

L-shape components: Our formulation handles both the rectangular and *L-shaped* components. An L-shaped component is modeled as a pair of abutting rectangles, as shown in Figure 4.

Orientation: Each component, including L-shaped components, can assume one of two possible orientations. The orientation which minimizes overall cost function will be automatically selected by the program.

Merging: Device merging is a widely used technique to improve both area and performance of custom designs. If two components with the same diffusion type share a common net, they can be combined in a single diffusion area. Merging, which has additional advantage of minimizing the number of placeable components, is handled automatically in our tool in a preprocessing step.

Sharing: In a typical organization of a datapath, horizontally adjacent cells are identical copies of each other, reflected with respect to the vertical boundary line. Component sharing is a technique that allows two component areas (diffusions or poly gates) that belong to components from adjacent cells to be merged, if they share the same *independent* net. Independent nets are the global control or clock signal nets which are carried by the horizontal bristles. In this case the components can be pushed beyond the cell boundary line to be shared with the component of the other cell. Our model automatically supports the component sharing.

C. Design Objectives

The primary objective is to minimize the layout area of the datapath cell. For a fixed datapath pitch, this is equivalent to minimizing the height of the cell. Performance and internal routability of the datapath components is considered as a secondary objective. It is achieved by minimizing some measure of interconnect complexity of the internal signal nets. In case of a single-well technology, the number of diffusion wells must also be minimized, as this improves yield and reduces the number of well plugs required. All these objectives are considered in our analytical model.

III. Approach to Datapath Design Automation

In order to manage the complexity of the transistor placement problem the entire design process is divided into the following phases: 1) initial relative placement, 2) component merging and grouping, 3) geometric placement, and 4) final post-processing and compaction.

A. Initial Relative Placement

The goal of this phase is to derive a relative initial placement of components that will facilitate modeling of the subsequent geometric placement problem. Specifically, initial relative placement provides an important information about relative values of X, Y coordinates of component centers which significantly simplifies the generation of non-overlapping constraints (see Section IV). Signal connectivity is the sole factor considered in this step, while component geometries are temporarily ignored. Standard force-directed technique [9] is used to compute the locations of the component centers, based on the connectivity of components to other components, bristles, and power rails. In order to avoid trivial solution the location of some components must be temporarily fixed. This is achieved by selecting a minimum number of fixed components, called *anchors*, and allocating then to each corner of the physical design space. Their location will be relaxed during the subsequent geometric placement, subject to the relative positions derived in the initial placement. Anchors are selected heuristically by analyzing the following factors: component area and geometry; connections to bristles, power rails and other components; and sharability across the cell boundaries. Fig. 2 shows a result of an initial relative placement, where components 1, 9, 2, and 10 have been selected as anchors.

B. Component Grouping

An important step in our procedure is the *grouping* of components into disjoint subsets (groups). This step is dictated by a need to limit the number of integer variables in our MILP formulation, described in Section IV-E. According to this formulation, an integer variable Q_{ij} represents a selection of a relative position for a pair of components (i, j). By combining components into groups, only one integer variable is needed to represent the relative position of all components in the group w.r. to the components in another group, hence significantly limiting the total number of variables needed. The components inside each group remain related to each other. In addition, we also apply component *merging*, described in Section II-B, which reduces the number of placeable components and further minimizes the number of integer variables.

Component grouping and merging is based on the initial relative placement, component connectivity, and electrical affinity (for example, the P and N transistors of an inverter can form a group). Its main goal is to limit the total number of integer variables so it can be efficiently handled by an MILP solver. Fig. 2 shows grouping and merging of components according to their initial placement and connectivity.

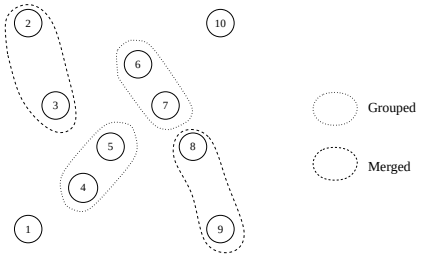

Fig. 2. Initial relative placement with component grouping

C. Geometric Placement

The goal of this phase is to generate non-overlapping placement of components, taking into consideration their geometries and physical design rules. The inter-component connectivity is addressed implicitly by maintaining the relative positions of the components determined in the first phase. The objective is to minimize the height of the cell layout. MILP model, described in the next section, is used to solve this problem efficiently.

D. Compaction

Compaction is a final post-processing step intended to reduce the number of P/N diffusion islands (if applicable) and to simplify routability. It may also help reduce the height of the layout [2].

IV. MILP Formulation of Geometric Placement

A. Component Modeling

Each rectangular component i is modeled as a pair of spatially related squares, whose coordinates of top/right corners are denoted by $(X_{i,1}, Y_{i,1})$ and $(X_{i,2}, Y_{i,2})$, as shown in Fig. 3. The width W_i and height H_i of component i are defined as its dimensions perpendicular and parallel to the poly line, respectively. The size of each squares is equal to $d_i = min(W_i, H_i)$. The coordinates of each component are defined by the coordinates of its first (top/right-most) square.An integer variable R_i is introduced for each component i to represent its *geometric orientation*. R_i=0 if the component is placed horizontally, and R_i=1, if it is placed

vertically (see Fig. 3). Square components do not require integer variables to represent their orientation. The following constraints

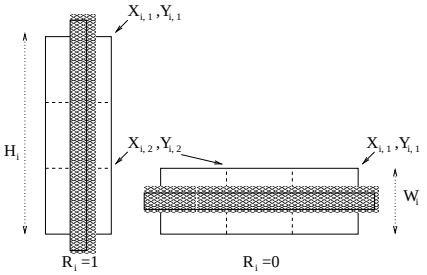

Fig. 3. Model of a component in two different orientations

properly model a rectangular component as a function of its orientation

$$X_{i,1} - X_{i,2} = (D_i - d_i) * (1 - R_i) \quad (1)$$
$$Y_{i,1} - Y_{i,2} = (D_i - d_i) * R_i \quad (2)$$

where $D_i = max(W_i, H_i)$ and $d_i = min(W_i, H_i)$.

B. Boundary Constraints

Each component must be placed within the boundaries of the cell, determined by its fixed pitch, X_0, and variable height, Y_0 (to be minimized). Let $\triangle_{bi}$ be a distance from the boundary of component i to the respective boundary of the datapath cell. The boundary constraints are given as follows:

$$X_{i,1} \leq X_0 - \triangle_{bi_x} \quad (3)$$
$$X_{i,2} \geq d_i + \triangle_{bi_x} \quad (4)$$
$$Y_{i,1} \leq Y_0 - \triangle_{bi_y} \quad (5)$$
$$Y_{i,2} \geq d_i + \triangle_{bi_y} \quad (6)$$

where $\triangle_{bi_x}, \triangle_{bi_y}$ are the required margins for the vertical and horizontal boundary lines, respectively. If component i is sharable across vertical reflection line, $\triangle_{bi_x}$ is negative to allow for intentional diffusion overlap. Otherwise $\triangle_{bi_x}$ is positive, and its value is determined by the respective spacing design rule.

C. L-shape Components

An L-shape component is modeled as a pair of rectangles attached to each other, with the same poly orientation. The *poly orientation* parameter P_i for component i is defined as follows:

$$P_i = \begin{cases} R_i & \text{if } H_i = D_i \\ 1 - R_i & \text{otherwise} \end{cases}$$

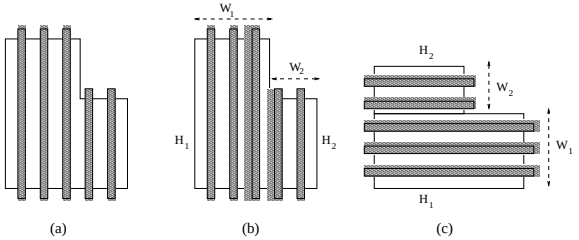

Fig. 4. Modeling of an L-shaped component: a) original component; b) component in vertical orientation, $P_i = 1$; c) component in horizontal orientation, $P_i = 0$

Since P_i and R_i are related, there is no need to define a new integer variable. The following constraints model the rotation of an L-shaped component [1].

$$X_{i,1} - X_{j,1} = W_i * P_j - (H_j - H_i) * (1 - P_j) \quad (7)$$
$$Y_{i,1} - Y_{j,1} = W_i * (1 - P_j) - (H_j - H_i) * P_j \quad (8)$$

where i and j are the smaller and bigger rectangle, respectively. In the above equations, P_j is used to force the same poly orientation for both rectangles. Refer to Fig. 4, where $i = 2, j = 1$, and $P_i = R_i$.

D. Folding and Components with Multiple Instances

Each component is allowed to assume several different shapes (instances), one of which is chosen in the final placement. For components with two instances, an integer variable S_i (*instance selection parameter*) is introduced for component i. The model handles multiple instances at a cost of adding new integer variables, needed to represent the selection and linearize the constraints. The details are omitted for simplicity.

E. Non-overlapping Constraints

To satisfy non-overlapping constraints, the distance between two components must be greater than or equal to the required diffusion spacing, specified by the design rules. Since the 1-D relative positions of the components are known from the initial placement phase (one for each physical dimension), these constraints can be written as a set of linear inequality constraints. An integer variable $Q_{i,j}$ is introduced for the each pair of components (i, j), to represent their relative placement. It determines whether the X or Y spatial relation is active. The non-overlapping constraints take the following form:

$$X_{j,2} - X_{i,1} \geq d_j * Q_{i,j} + L * (1 - Q_{i,j}) + \triangle_{sep} \quad (9)$$
$$Y_{j,2} - Y_{i,1} \geq d_j * (1 - Q_{i,j}) + L * Q_{i,j} + \triangle_{sep} \quad (10)$$

Here L is a sufficiently large positive number, and $\triangle_{sep}$ is the required spacing between the components. With this formulation, fixing variable $Q_{i,j}$ fixes the spatial relation between components i and j to either horizontal or vertical one. As shown in Fig. 5, $Q_{i,j}$=0 means that component j will be placed *above* component i, and for $Q_{i,j}$=1 component j will be placed *to the right* of component i.

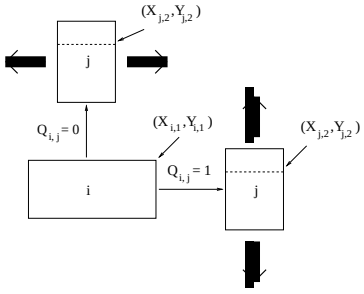

Fig. 5. Modeling the non-overlapping constrains with integer variables $Q_{i,j}$

F. Optimization Problem

The final objective is to minimize the cell height Y_0 subject to the set of linear constraints defined above. We solve this problem using a commercial integer linear programming tool.

[1] While there are eight different orientations for such a constructed L-shaped component, this mode allows for only two orientations.

G. Complexity Issues

A typical transistor placement problem for a datapath cell involves 20-50 original components (transistors or predefined black boxes). After component merging, the size of the typical problem reduces to less than 20 placeable components, and is further divided into groups. To make our approach computationally feasible, we impose limit on the number of integer variables in our MILP formulation. We observed that MILP can run efficiently (and generate result within several minutes) if it is limited to about 60 integer variables. This number is problem-specific and can vary for different optimization problems. We control the complexity of the underlying optimization problem in several ways, including the merging and grouping of components, discussed earlier in Section III-B. Both of these techniques effectively reduce the total number of integer variables to the desired level, allowing the optimization program to complete within a few minutes. We should emphasize that the placement is carried out for *all* components at once, while restricting relative positions of components by their group membership. When needed, the component grouping is relaxed and the optimization problem repeated with the relative positions ($Q_{i,j}$ variables) of all components *except* the ones related to the group in question, fixed. This is done iteratively during the final post-processing step, and often significantly improves the result.

V. Results

We implemented the analytical technique described in this paper as a set of loosely integrated experimental tools, including force-directed initial placement and MILP-based geometric placement.

We tested our programs on several datapath circuits made available by Compaq's Alpha Development Group. The complexity of those circuits ranged from 10 to 30 transistors, with the number of placeable components being 9 to 15. We compared our results with those obtained manually by an experienced layout designer. We are not aware of any other synthesis tool for custom datapath design for the purpose of fair comparison.

Table I gives the comparison of final (routed) results with the best results obtained manually. The cell height is given in λ. All generated layouts were routable but some required minor modifications to accommodate poly/metal contacts and to facilitate routing.

		Height [λ]		CPU time [min]
Circuit name	# of trans.	Custom design	Our approach	Our approach
PGK9	26	66	78.5	1:40
PB7	11	42	42	0:20
CS15	47	155.5	150	2:30
MU9	37	77.5	90	4:20

TABLE I
Results

It is important to emphasize that the manually generated results include additional techniques and all possible "tricks of the trade". Specifically, in manual designs it is allowed to custom fold transistors and components in the best possible way to fit them in the available layout slots, while our methods can only deal with the finite number of *predefined* component instances. This was the case with example MU9 in the table, where only one instance of each component was provided as input to our program.

Example PGK9 suffers from a poor choice of anchoring. The sensitivity of our method to anchoring heuristics is an important problem that needs to be addressed.

In the CS15 example, our automatically generated placement was routable after minor modifications involving placing poly and metal contacts required for routing. As seen in the table, our result compares favorably with the manual one, while taking only several minutes to compute.

All of our initial results are within acceptable area overhead, while the development time is significantly lower; they take just a few minutes per design, which compares favorably with the speed of about 10 components per day for manual designs.

VI. Conclusions

This paper addresses our first attempt at automating datapath layout design. It currently concentrates on transistor placement, ignoring for now the important routing issues. As a result, all examples required some form of modification to accommodate routing (mostly by means of providing space for poly/metal contacts). The purpose of this first round of experiments was to validate our analytical approach. Our approach is able to handle designs with up to 50 components in less than 10 minutes. We had impressive time reduction within acceptable limits of area overhead compared to manual designs. Our method seems to give superior results for larger circuits, where human limitations at handling large design complexity becomes apparent. We are now in the process of gathering and evaluating the invaluable feedback from the designers so we can incorporate it in the future editions of our program.

Future work in this project will focus on the several issues. 1) The most important one is to develop a systematic way to include routability measures in our analytical model. 2) We need to develop better anchoring heuristics. 3) The grouping concept needs to be further explored to help reduce the number of integer variables. 4) We will also examine iterative improvement to MILP solutions by means of iterative group relaxation as this will allow us to handle larger designs and obtain better results.

We are confident that our analytical approach combined with proper post-processing and iterative improvement will create automatically routable and fully acceptable layouts competitive with those obtained by human designers.

VII. Acknowledgement

Authors would like to thank Sam Levitin, Ken Slater and Alan Cave, of Compaq Alpha Development Group, for their invaluable feedback and support.

References

[1] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley-Teubner Series, 1990.

[2] D. Vahia and M. Ciesielski, "Transistor level placement for full custom datapath cell design," *International Symposium on Physical Design*, 1999.

[3] H. Onodera, Taniguchi Y., and Tamaru K., "Branch-and-bound placement for building block layout," *Proceedings of 28th Design Automation Conference*, pp. 433–439, 1991.

[4] C Cheng and E. S Kuh, "Module placement based on resistive network optimization," *IEEE Transactions on CAD*, pp. 218–225, 1984.

[5] C. Sechen, "Chip-planning, and global routing of macro/custom cell integrated circuits using simulated annealing," *ACM/IEEE Design Automation Conference*, 1988.

[6] J. Cohn, "KOAN/ANAGRAM II: New tools for device-level analog placement and routing," *IEEE Journal on Solid-State Circuits*, pp. 330–342, 1991.

[7] E. Charbon, "A constraint-driven placement methodology for analog integrated circuits," *Proceedings of CICC*, pp. 2821–2824, 1992.

[8] S. Sutanthavibul, E. Shragowitz, and J. B. Rosen, "An analytical approach to floorplan design and optimization," *IEEE Transactions on CAD*, pp. 761–769, 1991.

[9] N. A. Shervani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993.