Floorplanning for Mixed Macro Block and Standard Cell Designs *

Arun Shanbhag, SrinivasaRao Danda and Naveed Sherwani
Department of Computer Science
Western Michigan University
Kalamazoo, MI 49008

Abstract – *In this paper, we present ARCHITECT, the floorplanner for high performance Mixed Block and Cell designs. A novel and significant feature of ARCHITECT is that it exploits the flexibility of the standard cell regions by generating arbitrary rectilinear shapes for the flexible blocks.*

We have implemented ARCHITECT on a Sun SPARC station 1+ using C and Xview. We have tested the floorplanner on various randomly generated examples. Experimental results indicate that ARCHITECT generates floorplans with minimal white space within the user specified bounds.

1 Introduction

Floorplanning is a critical step in VLSI physical design and it is the process of determining the shapes and locations of different blocks of a chip. The floorplanning problem for a design consisting of only one type of blocks, either macro or flexible, with the shapes of flexible blocks restricted to rectangular, is referred to as the classical floorplanning problem. Several heuristic algorithms have been developed for the classical floorplanning problem (See [1], Chapter 5). Currently, many industrial chip designs have a mix of both macro blocks and standard cells and are called as *Mixed Block and Cell* (MBC) designs. The macro blocks are also called *fixed* blocks as their areas, shapes and pin locations are well defined. On the other hand, the standard cell blocks can be molded into a variety of rectilinear shapes. Hence these blocks are referred to as *juice* or *flexible* blocks. Typically, a chip consists of about 10 to 20 fixed blocks and about 5,000 to 10,000 standard cells. Our floorplanning problem is motivated by this new layout methodology.

In [2, 3, 4, 5] algorithms for floorplanning of MBC designs have been considered. All the existing floorplanning algorithms for the mixed block and cell designs restrict the standard cell block shapes to rectangular in order to simplify the problem at hand. However, high performance circuit layout needs can only be met by considering more general shapes. In [5], the pre-designed block shapes are considered to be rectilinear and the soft modules are considered to be rectangular with varying aspect ratios. However, the flexibility offered by the soft modules or standard cell blocks is not completely utilized. The consideration of rectilinear shapes for standard cell blocks not only leads to reduction in chip area but most importantly improves circuit performance as shown in Figure 1.

*Research supported in part by National Science Foundation Grant USE-9052346, and ACM/SIGDA Scholarship 1993.

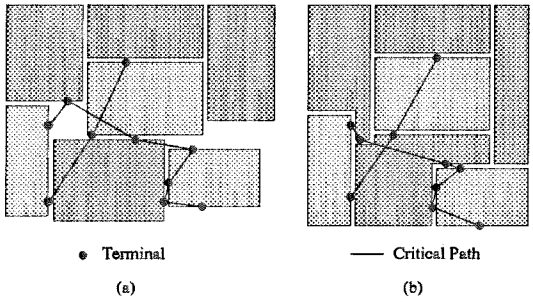

Figure 1: Reduction in critical path lengths due to rectilinear shape assignment for flexible blocks.

In this paper, we present ARCHITECT, a floorplanner for mixed block and cell designs. The most novel and significant feature of ARCHITECT is that it exploits the flexibility of the standard cell regions by generating arbitrary rectilinear shapes for the flexible blocks. The designer is provided with two *shape parameters* which can control the shapes of the flexible blocks. The shape parameters are (i)α, which controls the number of line segments in the boundary of a flexible block and (ii)β, which controls the distance between two parallel line segments defining the boundary of a flexible block. These parameters have to be defined for each block and can be used to vary the shape of the flexible blocks from rectangular to highly complex rectilinear shapes.

The rest of the paper is organized as follows. Section 2 describes the preliminary definitions, Section 3 gives an outline of our approach, Sections 4 and 5 discuss the various steps of ARCHITECT and the experimental results are discussed in Section 6.

2 Preliminaries

The input to the ARCHITECT system consists of a set of fixed blocks, $\mathcal{B}_p = \{b_1, b_2, \ldots, b_p\}$, a set of standard cells, $\mathcal{S} = \{s_1, s_2, \ldots, s_q\}$, a set of nets, $\mathcal{N}_p = \{n_1, n_2, \ldots, n_r\}$, a set of critical paths $\mathcal{C}_p = \{c_1, c_2, \ldots, c_s\}$ and the target chip width, W and height, H. Also given for each fixed block b_i is a three tuple (w_i, h_i, T_i^b), where, w_i is the width, h_i is the height and T_i^b is the terminal information. The terminal information $T_i^b = \{(t_1, x_1, y_1), (t_2, x_2, y_2), \ldots, (t_t, x_t, y_t)\}$, where t_i is a

unique terminal number, (x_i, y_i) give the coordinates of terminal, t_i, with respect to the lower left corner of b_i and t is the number of terminals for b_i. A standard cell, $s_i, 1 \leq i \leq q$, is represented by a 2-tuple (w_i, T_i^c) where w_i is the width of s_i and $T_i^c = \{(t_1, x_1, y_1), (t_2, x_2, y_2), \ldots, (t_u, x_u, y_u)\}$, is the set of terminals which belong to s_i, where t_u is a unique terminal number, (x_u, y_u) denote the coordinates of t_u with respect to the lower left corner of the standard cell and u is the number of terminals for s_i. We assume that all the standard cells belong to the same library and hence have the same height. A *net* n_i, $1 \leq i \leq r$, is a wire which connects a group of terminals given by $T_{n_i} = \{t_1, t_2, \ldots, t_v\}$, where each terminal, t_i, either belongs to a fixed block or a standard cell. Each critical path, $c_i = \{a_j | a_j \in \mathcal{B}_p$ or $a_j \in \mathcal{S}\}$, is a set of components through which the path passes starting from the input to the output.

3 Outline of Our Approach

The ARCHITECT floorplanning algorithm consists of four steps. Initially, the ARCHITECT system iteratively partitions the netlist $\mathcal{N}_p$ according to the partitioning strategy in [4], till each partition consists either of a single macro block or only standard cells. The next step is to generate an initial placement for these blocks. The objective of the Initial Placement (IP) algorithm is to find the orientation and placement for all the blocks such that the critical path lengths and the total wirelength is minimized. The initial placement generated can have some vacant or *white* spaces. The next step is the Flexible Block Reshaping (FBR) phase, in which we try to eliminate the white spaces by reshaping the flexible blocks. After the shapes of the flexible blocks are determined, we use the SOAP algorithm [6] for placing the standard cells within each flexible block. The generation of rectilinear shaped standard cell region does not necessitate the use of complex channel routers as in future designs channels will be completely eliminated because of the availability of more layers for routing.

4 Initial Placement

In this step an initial placement is generated for the blocks using an hierarchical approach. The main steps involved are generation of the placement tree, shape estimation at each node of the tree followed by the block orientation and placement phase. We now discuss briefly each step of the IP algorithm.

1. **Placement Tree Generation:** Initially all the blocks are represented as leaves of the tree and the clustering proceeds bottom-up. The leaves of the tree are at the lowest level of the tree, also called level 0, and the root is at the highest level. The level 1 parent nodes of the tree are generated by clustering atmost four blocks from the leaf level. The blocks at higher level (greater than 1) are generated by clustering atmost two blocks from the immediate lower levels. The blocks within a child node of the tree are called *child blocks* while the blocks represented by a parent node is called a *megablock*. In order to find, which blocks will be clustered to form a megablock at a given level of the placement tree, we consider the combination of every pair of blocks at that level. This set of combinations represented by $\mathcal{D} = \{S_1, S_2, \ldots, S_m\}$, where each S_i is a set of blocks and $|S_i| = 2$. The total weight of nets, w_i, between all pairs of blocks of set S_i, is calculated where $w_i = \sum_{\forall j \in S_i} \sum_{\forall k \in S_i, k \neq j} n_{kj}$ where, n_{kj} is the number of nets between blocks B_k and B_j($B_k, B_j \in S_i$). We denote a function $f(i,l) = 1$, if $(B_j, B_k) \subset c_l, 1 \leq l \leq v, B_j, B_k \in S_i$ and $f(i,l) = 0$, otherwise. Let $cb_i = \sum_{l=1}^{v} f(i,l)$. Let, $d_i = \frac{cb_i}{C_2^{|\cup_{1 \leq i \leq m} S_i|}}$, then the *clustering factor*, v_i for S_i is $v_i = W_n \times \frac{w_i}{|\mathcal{N}|} \times 100 + W_c \times \frac{c_i}{d_i} \times 100$, where W_n is the weightage assigned for connectivity and W_c is the weightage assigned for criticality. The blocks of set, S_i, which have the maximum clustering factor are clustered together to form a megablock. These blocks are not considered for clustering with other blocks at the same level. This procedure is carried out until all the blocks at the current level are clustered. After combining the pseudo blocks to generate a megablock, all the nets that are internal to the megablock are eliminated from the netlist as these nets need not be considered for megablock formation at the next higher level.

2. **Shape Estimation of Placement Tree:** The chip dimension ($W \times H$) provided as input, sets the target shape for the root node of the tree. The target shape at the root node is used to generate a target shape for the child nodes of the root node. This process is carried out in a top down fashion till each megablock in the placement tree has a shape allocated to it.

3. **Block Orientation and Placement:** In this step the blocks are oriented and placed within a megablock to achieve the target shapes estimated for each parent node by the shape estimation phase. This step consists of two phases. In the first phase, we find the orientations of the blocks relative to one another depending on the critical path and connectivity between the blocks. After the orientation of each block at a parent node has been determined, we find the exact position for each block within the shape allocated to the node in the shape estimation phase so that the megablock has a shape which is close to the bounding rectangle.

5 Improvement of Initial Placement

In this section we describe how the initial placement generated by IP algorithm is improved by the Flexible Block Reshaping (FBR) algorithm. Each block is processed by the FBR algorithm and is placed on the top boundary of the partial floorplan generated. The top boundary of the partial floorplan is a sequence of line segments, $L = \{l_1, l_2, \ldots, l_t\}$. The partial floorplan consists of all the blocks that have been processed by the FBR algorithm in the previous iterations. The FBR algorithm has the following steps.

1. **Straight Boundary Block Assignment:** Initially, when none of the blocks of the floorplan have been processed, the partial floorplan does

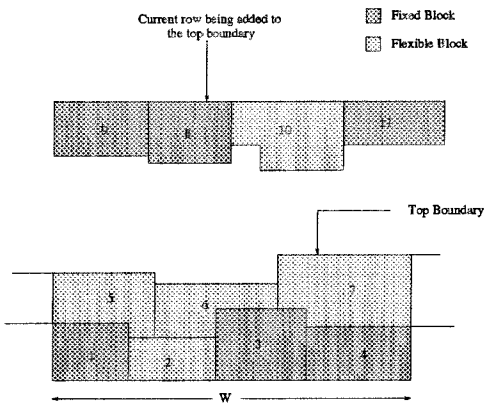

Figure 2: Floorplanning process showing the top boundary of the partial floorplan.

not consist of any block and the partial floorplan boundary is a straight line segment of width W. We process the first row of blocks together so as to assign them within width W on the partial floorplan boundary. The top boundary of the partial floorplan is stored as a sequence of iso-oriented line segments $L = \{l_1, l_2, \ldots, l_t\}$. The *type* of a line segment is defined as

$type(l_i) = h$, if the line segment is horizontal
$type(l_i) = v$, if the line segment is vertical

A vertical line segment which has an end point with the y-coordinate greater(less) than that of the immediate preceding horizontal line segment in L, is called a *rising vertical* (*falling vertical*) line segment and is said to be a v_r (v_f) segment. The line segments in L satisfy the following properties.

(a) **P1:** $type(l_i) \neq type(l_{i+1})$, $1 \leq i \leq t-1$.

(b) **P2:** $l_i \cap l_j = \phi$, $1 \leq i, j \leq t$ where the line segments l_i and l_j are horizontal and $\cup_{i=1}^{t} l_i = W$, the width of the chip.

(c) **P3:** The first and last line segments of the top boundary are always horizontal and number of horizontal line segments will be one more than number of vertical line segments.

We define δ as the vertical distance between the two line segments l_i and l_j such that $type(l_i) = type(l_j) = h$ and l_i has the maximum y-coordinate among all horizontal segments in L and l_j has the minimum y-coordinate among all horizontal segments in L. If a line segment $l_i \in L$ is of type v_f and the line segment l_{i+2} is of type v_r, then the line segments l_i, l_{i+1} and l_{i+2} are said to define a *depression* in the top boundary of the partial floorplan.

In SBBA, we are given,

(a) $\mathcal{B}_\pi = \{B_1, B_2^*, B_3, \ldots, B_n\}$, a set of rectangular blocks such that B_i^* is a flexible block and all other blocks are fixed blocks. Each block B_i has width w_i and height h_i and $\sum_{\forall i} w_i = W'$.

(b) W, the total width within which the blocks must be assigned to the partial floorplan boundary.

In this phase of FBR, we solve the SBBA problem. The objective is to assign the blocks to the partial floorplan boundary such that, δ is minimum subject to the constraint that the bounding rectangle enclosing all the blocks does not have a width greater than W and the permutation of the blocks given by $\mathcal{B}_\pi$ is not altered.

Let $h_{max} = \max(h_i)$, $\forall$i, such that B_i is a fixed block. We create a rectangular region, R, with width W and height h_{max} on the top boundary of the partial floorplan. If any fixed block lies outside the region R, then we move all these blocks so that they lie within R such that two fixed blocks B_i and B_j are separated by atleast β if there exists a flexible block B_k^* which lies in between B_i and B_j in $\mathcal{B}_\pi$. If there are m fixed blocks each with height h_{max}, then they divide the region R into $m+1$ subregions. We assign the locations for the fixed blocks on the partial floorplan boundary such that, $Area(R_i) = \sum_{\forall i, B_i \in R_i} Area(B_i), \forall R_i$. Once these fixed blocks are assigned and the regions are fixed, we assign shapes for the flexible blocks within each region R_i to fit the region.

2. **Irregular Rectilinear Boundary Block Assignment (IRBBA):** After the first row of blocks is placed, the top boundary of the partial floorplan may not be a straight line. It may consist of a set of horizontal and vertical line segments. Hence the problem for assigning the next row of blocks is to place the set of blocks such that δ is minimum and the bounding rectangle for these blocks has a width no more than W. In IRBBA in addition to $\mathcal{B}_\pi$ and W, we are also given, L, a list of line segments consisting of horizontal, rising vertical and falling vertical edges which define the top boundary of the partial floorplan. The objective for IRBBA problem is the same as that for SBBA problem.

   In this case, we first assign all the fixed blocks to the top boundary of the partial floorplan and then use the remaining area to reshape the flexible blocks. The objective for assigning the fixed blocks is to keep the height of the bounding for the fixed blocks to a minimum without permuting the blocks. The width of the bounding rectangle is no more than W. The fixed blocks are assigned to the best possible depression in the top boundary so that the height of the bounding rectangle can be kept to a minimum. We then scan the bounding rectangle from left to right to assign shapes for all the flexible blocks.

3. **Irregular Rectilinear Boundary Block Assignment with Straight Top (IRBBAST):** The problem of assigning the last row of blocks

which form the top boundary of the floorplan is different from problem IRBBA. This is because, after assigning the blocks to the top boundary, the new top boundary generated must be a straight line. Hence the problem of assigning the last row of blocks is to place the set of blocks such that the increase in height of the chip is minimized and the bounding rectangle for these blocks has a width no more than W. In IRBBAST we are given, $\mathcal{B}_\pi$, W and L.

The objective is to assign the blocks to the partial floorplan boundary such that, increase in height of the chip is minimum subject to the constraint that the bounding rectangle enclosing all the blocks does not have a width greater than W and the permutation of the blocks given by $\mathcal{B}_\pi$ is not altered. The fixed blocks, if any, in $\mathcal{B}_\pi$, are assigned locations such that $x_i + h_i = h_{max}$, $\forall i$ and $B_i \in \mathcal{B}_\pi$ where B_i is a fixed block and h_{max} is minimum. The bounding rectangle for these blocks has a width no more than W. After assigning the fixed blocks, the flexible blocks are reshaped by scanning the region from the top to the bottom instead of from left to the right as earlier. This ensures that the floorplan has a straight top boundary.

After all the blocks are processed, the layout is rotated by 90° and the above process is repeated in the other direction. The floorplan is iteratively compressed till no significant reduction in area is achieved.

6 Experimental Results

The ARCHITECT system has been implemented in C, using Xview on a SPARC station 1+. It was tested on randomly generated examples. Table 1 gives the chip area of the various examples that were tested along with the number of fixed blocks, standard cells and the number of nets.

The FBR algorithm drastically reduces white space in the floorplan. The reduction in white space depends on the parameters α and β set by the user. It also depends on the relative number of fixed blocks and standard cells in the layout. The layout areas indicated in Table 1, are for $\alpha = 5$ and $\beta = 30$ units. Usually, the FBR algorithm requires 3 to 4 iterations to converge to the best layout possible with the given values of α and β. Notice that for F4, the layout has minimum amount of white space. This is due to the large number of standard cells available in the layout. As the number of blocks increases and the number of standard cells is reduced (F5) the white space in the floorplan increases. We have noticed that initial size and shape of the blocks have influence on the initial placement but the final floorplan is independent of the initial block sizes.

Figures 3 shows the output of the Initial Placement algorithm and the Flexible Block Reshaping Algorithm for example F2. Figure 3(a) is the initial floorplan generated by the IP algorithm for example F2. Figure 3(b) shows the final layout generated by the FBR algorithm with $\alpha = 5$ and $\beta = 30$, for all the blocks. The width and height requirements were set to 220 units each. Figure 3(c) shows the final layout generated by the FBR algorithm with $\alpha = 8$ and $\beta = 15$ units. As we relax the values of α and β, the amount of white space in the final layout decreases. Notice that fixed blocks 7 and 9, which had 5 edges in Figure 3(b) have 8 edges in Figure 3(c). Relaxing the value of β eliminated the white space (Figure 3(c)) which was enclosed by blocks 12, 3, 15 and 14 and by blocks 1, 7, 12 and 4.(Figure 3(b)).

No.	blks	cells	nets	A_c†	A_f‡	% WS*
F1	3	800	400	40664	44204	8.01
F2	5	900	600	51242	55194	7.17
F3	8	1400	890	75947	83458	9.0
F4	12	5000	1400	221941	234858	5.5
F5	15	1200	750	104153	118356	12.0
F6	10	2200	750	101032	108636	7.0

Table 1: Floorplan areas generated by ARCHITECT. †-Area of blocks & cells, ‡-Floorplan Area, *-White space

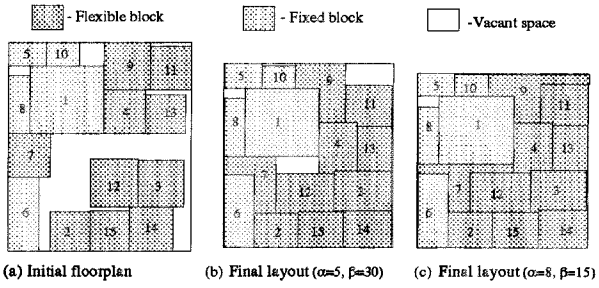

Figure 3: Layout of F2.

References

[1] Sherwani, N. "Algorithms for Physical Design Automation", Kluwer Academic Publishers, 1992.

[2] C. Sechen, "Chip-Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing". *Proc. 25th Design Automation Conference*, 1988, pp. 73-80.

[3] Upton, M., Samii, K., and Sugiyama, S., "Integrated Placement for Mixed Macro Cell and Standard Cell Designs", *Proc. of 27th ACM/IEEE Design Automation Conference*, 1990, pp. 32-35.

[4] Apte, J. and, Kedem, G., "Heuristic algorithms for combined standard cell and macro block layouts", *Proc. of the Sixth MIT Conf. on Advanced Research in VLSI*, 1990, pp. 367-385.

[5] Tsu-chang Lee, "A Bounded 2D Contour Searching Algorithm for Floorplan Design With Arbitrarily Shaped Rectilinear and Soft Modules", *Proc. of the 30th ACM/IEEE Design Automation Conference*, 1993, pp., 525-530.

[6] Sung-Soo Kim and Chong-Min Kyung, "Circuit placement on arbitrarily shaped regions using self-organization principle", *IEEE Trans. on CAD*, Vol.11, No.7, pp.844-854, July 1992.