An Analytical Approach to Floorplan Design and Optimization.

Suphachai Sutanthavibul, Eugene Shragowitz, J. Ben Rosen

Computer Science Department
University of Minnesota
Minneapolis, Minnesota 55454

Abstract

An analytical method for general floorplan design and optimization is proposed. This method is based on a mixed integer programming model and application of a standard mathematical software. The method allows arbitrary combinations of rigid and flexible modules. Various objective functions such as chip area, interconnection length, timing delays or any combinations of them are permitted. Routing space is estimated by the global router. Experimental data are provided.

1. Introduction

Floorplanning is one of the first steps of VLSI design. It is assumed, that at this step rough estimations of the modules areas and interconnections between them are known, while topology of the chip and exact dimensions of the modules are get to be defined. Common goals in the floorplan design are minimization of the chip area, minimization of interconnection length, or combination of both. Another common problem is optimization of the floorplan area for the given topology. A method described in this paper covers all formulations of the floorplanning problems mentioned above without restricting a class of solutions to the slicing structures. In addition, it allows consideration of realistic situations when some modules have rigid shapes and the remaining modules have flexible shapes, or all modules have fixed dimensions, i.e., it can be applied to solving placement problems.

Like other methods proposed for such classes of problems, the proposed method provides a suboptimal solution. This suboptimal solution is obtained by the consecutive solutions of the subproblems of the smaller size. Unlike many other methods our method obtains the optimal solution for each subproblem. Each subproblem is solved as a 0-1 mixed integer programming problem. The final solution is achieved by successive augmentation of new elements to the already constructed partial solution. In the process, the partial solution is reformulated to reduce the number of variables in the mixed integer programming problem. That makes it possible to keep the number of variables close to a constant in each step of the process. The timing complexity of this method is linear with respect to the number of partitions. This allows solution of the floorplanning problems of any practical dimensions. Benchmarks constructed for the Workshop on Physical Design, organized by MCNC in May 1988, are used to illustrate the performance of this methodology.

The text of this paper is structured as follows. Section 2 describes the mixed integer programming formulation of floorplanning. The solution method is described in section 3. Experimental results are given in section 4.

2. Mixed Integer Programming Model of Floorplanning

2.1. Previous Works

Starting from the early seventies a substantial number of papers on floorplanning have been published. Several fundamental papers on floorplan design belong to Otten [OTT82, OTT83]. Starting from Otten, almost all authors relied on the slicing structures for producing floorplans. Our methodology does not assume application of slicing. Among the most recent work based on the slicing approach is a paper by Wong [WON86]. Wong presented a floorplan design algorithm using Simulated Annealing.

Mueller et al [MUE87] described a bottom-up iterative algorithm for the floorplan design. The floorplan representation used in this algorithm is also based on the *normalized Polish expressions* and slicing structures. Another direction in floorplanning was introduced by Heller et al. [HEL82]. It is rectangular dualization and rectangular dissection. This technique was developed by Kozminski, Bhasker and others [KOZ84, BHA86, KOZ88].

A mixed integer programming model of placement differing from ours was proposed in [MAR84]. Several transformations of the original placement problem were made there. For example, any rectangular block with the aspect ratio that is less than or equal to a user-specified number is replaced by a group of square blocks with an edge size equal to the minimal edge of the original block. It is difficult to judge the effectiveness of this technique because only one simple example with three blocks was provided. Unlike that model, our model does not require the simplifications mentioned. In addition, we are using a linear mixed integer programming model instead of nonlinear. Our solution technique for the mixed integer programming problem allows us to solve placement problems with a substantial number of blocks.

2.2. Problem Definition

Given a set of K_1 flexible modules and K_2 rigid modules, $K_1 \bigcup K_2 = K$. For each flexible module i ($i \in K_1$) area S_i is defined, by $w_i h_i = S_i$, where w_i and h_i are unknown width and height of the module i. Constraints are imposed on the acceptable aspect ratios for each flexible module. For each $i \in K_1$ the inequality $b_i \leq w_i/h_i \leq a_i$ is presented. For rigid modules ($i \in K_2$) w_i and h_i are given and it is assumed that the 90° rotation of modules is allowed. This is equivalent to conditions $a_i \leq w_i/h_i \leq b_i$ or $1/a_i \geq w_i/h_i \geq 1/b_i$. The connectivity information is given in the form of the *netlist*, where for each module a set of nets incident to the module is presented. This allows us to define a number of common nets c_{ij} for each pair of modules i and j. We define a floorplan by positions of the lower left corner (x_i, y_i) of each module in the system of coordinates with the center in the lower left corner of the chip. Nets are assigned to routing channels and estimations of channel widths are performed. Additional constraints on the length of critical nets can also be presented. Input information includes the widths and spacings of metals for routing in both horizontal and vertical directions. This information is technology dependent. The optimal floorplan corresponds to a minimal covering rectangle R of a set of K nonoverlapping rectangles representing modules and spaces between them reserved for interconnections.

2.3. Mixed Integer Programming Formulation

Let us consider the constraints preventing the overlapping of any pair of rectangular modules i and j. Let (w_i, h_i), (w_j, h_j) (assumed to be known at this point) denote the dimensions (width and height) of modules i and j respectively. Let (x_i, y_i), and (x_j, y_j) (unknown continuous variables) denote the position of the lower left corners of the modules i and j. To prevent overlapping of modules i and j, it is required that at least one of the following linear inequalities holds:

$$\begin{array}{lll} x_i + w_i \leq x_j & (i \text{ is to the left of } j) \\ x_i - w_j \geq x_j & (i \text{ is to the right of } j) \\ y_i + h_i \leq y_j & (i \text{ is below } j) \\ y_i - h_j \geq y_j & (i \text{ is above } j) \end{array} \qquad (1)$$

In order to ensure that at least one of these inequalities always holds we introduce two additional 0-1 integer variables x_{ij} and y_{ij}, which take only 0 and 1 values. Let us define bounding functions W and H, such that we always have $|x_i - x_j| \leq W$ and $|y_i - y_j| \leq H$.

Possible choices for W and H are: $W = W_{max}$(maximal width of the chip allowed) and $H = H_{max}$(maximal allowed height of the chip). If W_{max} and H_{max} are not known, then $W = \sum_K w_i$ and $H = \sum_K h_i$ can serve the purpose. Now consider the following system of linear inequalities for any pair of modules i and j:

$$\begin{array}{l} x_i + w_i \leq x_j + W(x_{ij} + y_{ij}) \\ x_i - w_j \geq x_j - W(1 - x_{ij} + y_{ij}) \\ y_i + h_i \leq y_j + H(1 + x_{ij} - y_{ij}) \\ y_i - h_j \geq y_j - H(2 - x_{ij} - y_{ij}) \end{array} \qquad (2)$$

It is easy to see that for each of the four possible choices of values for the integer variables, $(x_{ij}, y_{ij}) = (0, 0), (0, 1), (1, 0), (1, 1)$ only one of the four inequalities in (2) is active. For example, with $x_{ij} = 0$, $y_{ij} = 1$, only the third constraint applies, which allows the module i to be anywhere below the module j. The other three constraints are always satisfied for any permitted values of (x_i, y_i) and (x_j, y_j). Let us assume that one dimension of the chip is known, say W, and we need to determine a floorplan, which corresponds to the minimal height. Then we can impose the following additional constraints.

$$\begin{array}{ll} x_i \geq 0; \quad y_i \geq 0; & i \in K \\ x_i + w_i \leq W; & i \in K \\ \\ y^* \geq y_i + h_i, & \end{array} \qquad (3)$$

where y^* is the height to be minimized.

Then the first mixed integer programming formulation of the floorplanning problem for a case where all modules are rigid can be formulated as follows: *find minimum y^* subject to constraints (2) and (3).*

This mixed integer programming formulation requires $2K$ continuous variables, $K(K-1)$ integer variables, and $2K^2$ linear constraints. For example, if the number of modules $K=25$, then the number of continuous variables is equal to 50, the number of integer variables is equal to 600, and the number of linear constraints is equal to 1250.

It is obvious, that a better floorplan can be achieved if rotation of the rigid blocks is allowed. Our model can be easily adjusted to such extension. Let us introduce for each module i an additional 0-1 integer variable z_i, where $z_i = 0$ when rigid module i is placed in its initial orientation and $z_i = 1$ when module i is rotated 90°. The constraints of non-overlapping for two rigid modules can be rewritten in the following form:

$$\begin{array}{l} x_i + z_i h_i + (1 - z_i) w_i \leq x_j + M(x_{ij} + y_{ij}) \\ x_i - z_j h_j - (1 - z_j) w_j \geq x_j - M(1 - x_{ij} + y_{ij}) \\ y_i + z_i w_i + (1 - z_i) h_i \leq y_j + M(1 + x_{ij} - y_{ij}) \\ y_i - z_j w_j - (1 - z_j) h_j \geq y_j - M(2 - x_{ij} - y_{ij}) \end{array} \qquad (4)$$

where: $M = max(W, H)$

To keep the rotated rectangles within chip boundaries, the constraints (3) have to be rewritten as:

$$\begin{array}{ll} x_i \geq 0; \; y_i \geq 0; & i \in K \\ x_i + (1 - z_i) w_i + z_i h_i \leq W; & i \in K \\ \\ y^* \geq y_i + h_i, & \end{array} \qquad (5)$$

where y^* is the height to be minimized.

This initial model does not allow flexible modules, does not consider routing space and connection lengths, and requires substantial numbers of variables and constraints for an optimal solution. All these problems will be addressed later in the following sub-section.

2.4. Flexible Module in Integer Programming Formulation

The flexible model allows w_i and h_i to vary, but requires that the area S_i remain fixed, that is $w_i h_i = S_i$. We linearize this nonlinear relation about the point w_{imax}, to give h_i as a linear function of w_i (see Figure. 1) by applying the first two members of the Taylor series:

$$h_i = \frac{S_i}{w_{imax}} + \left(w_{imax} - w_i\right)\frac{S_i}{w_{imax}^2}; \qquad (6)$$
$$h_i = h_{i0} + \Delta w_i \lambda_i;$$

where: $h_{i0} = \frac{S_i}{w_{imax}}$; $\lambda_i = \frac{S_i}{w_{imax}^2}$; $\Delta w_i = w_{imax} - w_i$.

Therefore we need one additional continuous variable Δw_i for each flexible module. Then the condition of non-overlapping of flexible module i and rigid module j can be rewritten in the following manner:

$$\begin{array}{ll} x_i + w_{imax} - \Delta w_i \le x_j & (i \text{ is to the left of } j) \\ y_i + h_{i0} + \Delta w_i \lambda_i \le y_j & (i \text{ is below } j) \\ x_i - w_j \ge x_j & (i \text{ is to the right of } j) \\ y_i - h_j \ge y_j & (i \text{ is above } j) \end{array} \qquad (7)$$

To make only one of these constraints active two 0-1 integer variables x_{ij}, y_{ij} should be introduced for each pair of modules. A system of constraints emerging in this situation is very similar to the one described by the system of constraints (2).

If both modules i and j have flexible shapes, then conditions of non-overlapping will look as follows:

$$\begin{array}{ll} x_i + w_{imax} - \Delta w_i \le x_j & (i \text{ is to the left of } j) \\ y_i + h_{i0} + \Delta w_i \lambda_i \le y_j & (i \text{ is below } j) \\ x_j + w_{jmax} - \Delta w_j \le x_i & (i \text{ is to the right of } j) \\ y_j + h_{j0} + \Delta w_j \lambda_j \le y_i & (i \text{ is above } j) \end{array} \qquad (8)$$

The same technique as in (4) is used to make only one of these constraints active.

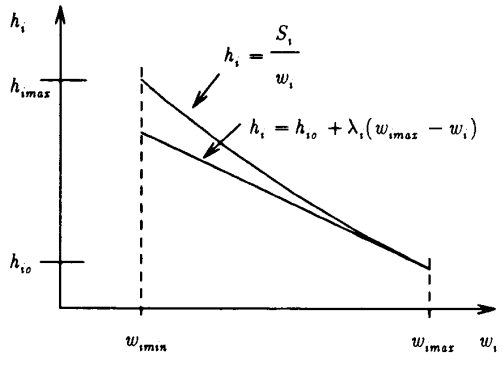


Figure 1.

2.5. Optimization of the Floorplan With the Given Topology

One of the often mentioned formulations of the floorplanning problem assumes that the topology of the chip is given and only shapes of the modules should be optimized. When the mixed integer programming formulation is applied to this problem, it results in elimination of all integer variables. These variables will assume values 0 or 1 according to the mutual positions of modules. It means, that it is known for any two modules i and j which one of them is on the top of another, or to the right of another. For example, if module j is on the right from module i according to the given topology, then $x_{ij} = 0$, $y_{ij} = 0$ and only one inequality $x_i + w_i \le x_j$ is needed. As a result, the number of integer variables for this formulation is equal to zero. The number of continuous variables is $2K$ and the number of linear constraints is $O(K)$ with the coefficient dependent on the average number of neighbors for each module for the given topology. Any standard linear programming software can be applied to this problem.

3. Solution Method for Mixed Integer Programming Model of Floorplanning

The solution for the mixed integer linear programming models described in the previous sections can be obtained by using standard mathematical programming software. There are several software packages available. We applied the widely used LINDO [SCH82] package for this purpose. This version of LINDO allows us to introduce up to 200 0-1 integer variables and a few thousand linear constraints.

The solution time for the integer programming problems grows exponentially (in the worst case) with the number of integer variables. In practical cases this growth is not exponential, but still can be very fast. This dictates a necessity for a technique that will consider only a limited number of modules (say 10-12) at a time. This technique is called successive augmentation (see Figure 2). The main idea of the successive augmentation is to create a final floorplan by adding a new group of modules to a partial floorplan in an optimal way until all modules are positioned. The successive augmentation does not guarantee an optimal solution for the original mixed integer programming problem. It guarantees optimality at each step of the process, resulting in a suboptimal solution of the original problem. This method represents the generalization of the greedy idea from one variable to a group of variables. Application of the successive augmentation in conjunction with the mixed integer programming requires solutions for several sub-problems.

1) How to select a new group of modules to be added to the partial floorplan?
2) How to minimize the number of integer variables for each solved integer programming problem?

These questions will be answered in the following subsections.

3.1. Covering Rectangles for the Partial Floorplan

The number of integer variables in the problem depends on the number of modules. Their mutual positions should be described by constraints. One way of reducing the dimensionality of the integer programming

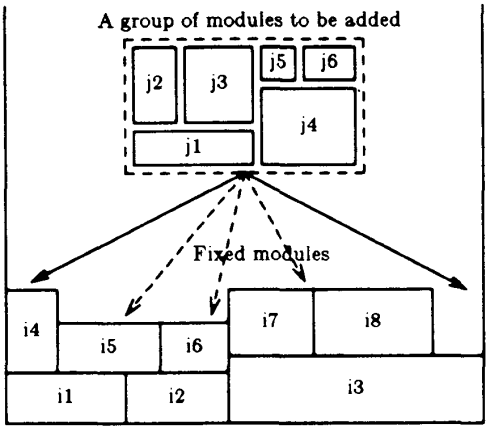


Figure 2. Floorplanning by Successive Augmentation.

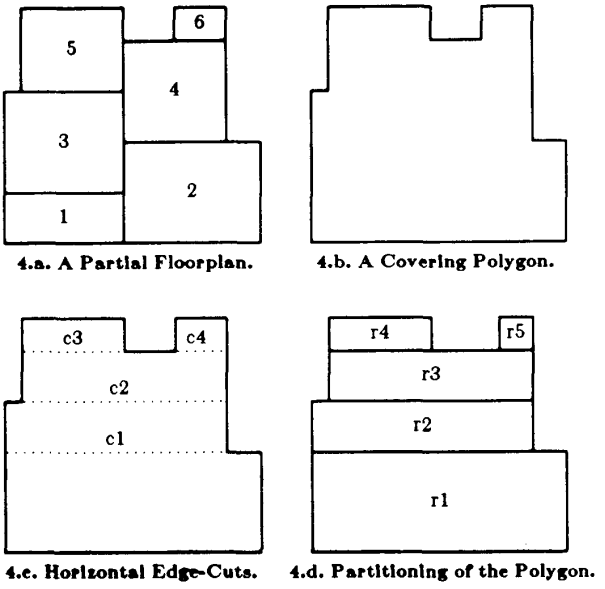
Procedure *FloorplanDesign*;
begin
(1) Select a group of m modules as a seed;
(2) Formulate a system of linear constraints for these unpositioned modules;
(3) Call an integer programming procedure to obtain the first partial floorplan;
(4) **while** ($m < k$) **do** { k is the total number of modules}
(5) Select a new group of e modules based on the connectivity to the already fixed modules in the partial floorplan and timing considerations;
(7) Find a set of d covering rectangles for the partial floorplan, where $d \leq m$;
(8) Formulate a system of linear constraints for d covering rectangles and e unpositioned modules;
(9) Call an integer programming procedure to obtain the partial floorplan;
(10) $m = m + e$;
(11) **endwhile**
(12) Perform global routing;
(13) Adjust floorplan;
end; /* Procedure *FloorplanDesign* */

Figure 3. An Algorithm for Floorplan Design.

problem is the replacement of already positioned modules by a set of covering rectangles. If the number of covering rectangles produced is less than the number of modules replaced, then reduction of the number of integer variables is achieved and the execution time is reduced. We propose an algorithm which generates a set of covering rectangles for the partial floorplan. An idea of this algorithm is illustrated by Figure 4.

A set of six fixed modules of Figure 4.a can be represented as a hole-free polygon in Figure 4.b. This polygon is then partitioned in the horizontal direction as shown in Figure 4.c. A set of five covering rectangles of this polygon is presented in Figure 4.d.

The purposed algorithm guarantees that the number of the covering rectangles produced by decomposition of



4.a. A Partial Floorplan. **4.b. A Covering Polygon.**

4.c. Horizontal Edge-Cuts. **4.d. Partitioning of the Polygon.**

Figure 4.

the polygon is always less than or equal to the number of fixed modules forming the polygon. In our case, the polygon of the partial floorplan always has a special feature - a flat bottom. Holes at the bottom of the polygon are ignored because new modules are added only from the open side of the chip. Some properties of this procedure are given below.

Let N be the number of the fixed modules in the partial floorplan and n be the number of horizontal edges of the polygon covering the partial floorplan. A horizontal edge-cut is a horizontal line that comprises at least one horizontal edge and it cuts from the polygon one rectangle.

Theorem 1: $n \leq N+1$

Proof: There are $2N$ horizontal edges for a set of N rectilinear modules. According to the floorplan design procedure, each module is placed either on the bottom line of the chip or on top of another module. Then, the bottom edge of each module is overlapping with the top edge of the nodule below, except, of course, of the modules placed on the bottom of the chip. The overlapping edges are merged into one edge which is a union of component edges. The number of horizontal edges produced by such a procedure is at most $2N - (N-1) = N+1$. The number of horizontal edges n of the covering rectangle can not exceed this number.

Let N^* be the number of partitions produced by the procedure *PartitioningPolygon*.

Theorem 2: $N^* \leq n - 1$

Proof: The procedure *PartitioningPolygon* starts from the bottom of the chip and produces a cut line for the first from the bottom horizontal edge of the covering polygon. This procedure separates from the polygon a rectangle whose bottom edge is the bottom of the chip and the top

edge is an edge-cut line. This process is repeated for each horizontal edge with an increased y-coordinate. Because this process starts not from the bottom but from the first horizontal edge with $y > 0$, then at most $n-1$ steps will be executed and $n-1$ rectangles are generated. Therefore, $N^* \leq n-1$.

Corollary: $N^* \leq N$

Proof: From $n \leq N+1$ then from $N^* \leq n-1$ it follows that $N^* \leq N$.

A further reduction can be achieved if a set of overlapping partitions is used instead of the nonoverlapping partitions to cover the partial floorplan.

3.2. Global Routing and Final Chip Area Computation

It is clear that a workable floorplan should allocate an area for interconnections. Different floorplans should be compared, not by the area produced by close packing of modules, but by the area, which includes modules and space occupied by routing. In our floorplanning methodology we assume, that the preliminary assignment of pins to sides of the modules is known (but without identifying exact locations of pins). This allows consideration of one generalized pin on each side of the module. Thus, instead of considering a center of a module as a generalized pin position we consider four generalized pins, one on each side. This model provides a more realistic evaluation of the routing space required for interconnections. The next important question is how to define positions and widths of routing channels. In our methodology this problem is solved in several steps. As the first step, we include the estimated area for interconnections in the objective function of the integer programming problem. The minimized objective is a combined area of the modules and interconnections. In addition, each module is placed into an envelope, which exceeds the initial size of each side by the value proportional to the number of pins on this side.

For example, if ten pins were assumed on the top side of the module and two pins on the bottom side then, an envelope exceeds the vertical dimension of the module on the value $12\mu_\lambda$, and the widths of the channels on the top and on the bottom would be $10\mu_\lambda$ and $2\mu_\lambda$ respectively, where μ_λ is the metal width plus spacing for one routing track in horizontal direction. These envelopes are used to solve a topological problem for the partial floorplans by an integer programming technique. In the next step of the algorithm global routing is performed for the system of channels defined by envelopes. Our global router is graph based. It uses the channel position graph obtained from the floorplan produced by the integer programming step and assigns a preliminary capacity to each edge defined by the envelopes. It uses the shortest path algorithm to find a route between two generalized pins. It also uses a penalty function for utilization of a channel beyond its preliminary capacity. Nets with the tight timing requirements are routed first [YOU89]. On the final step of the algorithm widths of channels are adjusted to accommodate results of the global routing and the final chip area is computed.

4. Experimental Results

Our methodology of floorplanning was implemented in a FORTRAN77 program running on Apollo DN3550 workstations (4 mips). It performs periodic calls to the LINDO [SCH82] package, which runs on the same computer. LINDO is executed as a procedure. For easy comparison of our results with those published earlier, a benchmark provided by the Workshop on Physical Design 1988, has been used. This benchmark, ami33, includes 33 modules. Several series of experiments were conducted.

	K	Modules Area	Chip Area	Area Utilization	Execution Time(mins)
n15	15	5057	5250	96.3%	1:50
n20	20	6870	7174	96.2%	2:55
n25	25	8575	8920	96.1%	4:14
ami33	33	11518	12353	93.3%	6:15

Table 1.

Series 1. The first series of experiments was conducted to study the influence of the problem size on the execution time of our floorplanner. Problems with 15, 20, and 25 modules were randomly generated and accompanied by the benchmark with 33 modules. Chip area was used as an objective function for the integer programming. Results of these experiments are given in Table 1. They indicate that execution time grows almost linearly with the problem size. This allows us to apply this floorplan to problem of any realistic dimensions.

Series 2. These experiments were conducted using the benchmark ami33 (total modules area is 11520). It was assumed that the technology requires over the cell routing. Two different objective functions were used for generating floorplans:

1. Chip Area.
2. Chip Area + Wire Length.

Two different algorithms were used for selecting the order in which modules were added to the partial floorplans.

1. Random.
2. Linear ordering based on connectivity.

The results of these experiments are assembled in Table 2. The best results achieved by this series corresponds to a chip utilization of 96%. A floorplan produced by these experiments is given in Figure 6.

	Linear Ordering		Random Ordering	
Objective Function	Chip Area	Wire Length	Chip Area	Wire Length
Chip Area	12588	3386	11976	3636
Chip Area + Wire Length	13716	3345	13176	3392

Table 2.

Series 3. The last series of experiments were conducted for a technology with around the cell routing. The same benchmark, ami33, was used. The objective functions for this series were taken from the same two classes as for series 3. Two techniques were used for providing routing area.

1. Floorplan Adjustment without Envelopes.
2. Floorplan Adjustment with Envelopes.

Two routing algorithms were applied.

1. Shortest Path.
2. Weighted Shortest Path.

The experimental results are given in Table 3. The wire length was measured based on the shortest paths produced by the global router. The results support our prediction that the application of envelopes allows us to decrease the chip size.

	No Envelopes		Envelopes	
Routing Method	Chip Area	Wire Length	Chip Area	Wire Length
Shortest Path	24101	5636	22378	5667
Weighted Shortest Path	23544	5948	21952	5978

Table 3.

REFERENCES

[BHA86] J. Bhasker, S. Sahni, "A linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph", Proc. 23rd Design Automation Conference, pp. 108-114, 1986.

[HEL82] W.R. Heller, G. Sorkin, K. Maling, "The Planar Package Planner for System Designers", Proc. 19th Design Automation Conference, pp. 253-260, 1982.

[KAN83] S. Kang, "Linear Ordering and Application to Placement," 20th Design Automation Conference, pp.457-464, 1983.

[KOZ84] K.A. Kozminski, E. Kinnen, "An Algorithm for Finding a Rectangular Dual of a Planar Graph for Use in Area Planning for VLSI Integrated Circuits", Proc. 21st Design Automation Conference, pp. 655-656, 1984

[KOZ88] K.A. Kozminski, E. Kinnen, "Rectangular Dualization and Rectangular Dissections", IEEE Trans. on Circuits and Systems, Vol. 35, No. 9, pp. 1401-1416, Nov 1988.

[MAR84] L. Markov, J. R. Fox and J. H. Blar, "Optimization Techniques for Two-Dimensional Placement", 21th Design Automation Conference, pp. 652-654, 1984.

[MUE87] T. Mueller, D.F. Wong, C.L. Liu, "An Enhanced Bottom-Up Algorithm for Floorplan Design", Proc. 24th Design Automation Conference, pp. 524-527, 1987

[OTT82] R.H.M. Otten, "Automatic Floorplan Design", Proc. 19th Design Automation Conference, pp. 261-267, 1982.

[OTT83] R.H.M. Otten, "Efficient Floorplan Optimization", Proc. International Conference on Computer Aided Design, pp. 499-502, 1983.

[SCH82] L. Schrage, "LINDO: Linear Interactive Discrete Optimizer", Copyright 1982.

[WIM88] S. Wimer, I. Koren, I. Cederbaum, "Optimal Aspect Ratios of Building Blocks in VLSI", Proc. 25th Design Automation Conference, pp. 66-72, 1988

[WON86] D.F. Wong, C.L. Liu, "A New Algorithm for Floorplan Design", Proc. 23rd Design Automation Conference, pp. 101-107, 1986

[WON89] D.F. Wong, P.S. Sakhamuri, "Efficient Floorplan Area Optimization", Proc. 26th Design Automation Conference, pp. 586-589, 1989.

[YOU89] H. Youssef, E. Shragowitz, L.C. Bening, "Critical Path Issue in VLSI Designs", Proc. International Conference on Computer Aided Design, pp. 520-523, 1989.

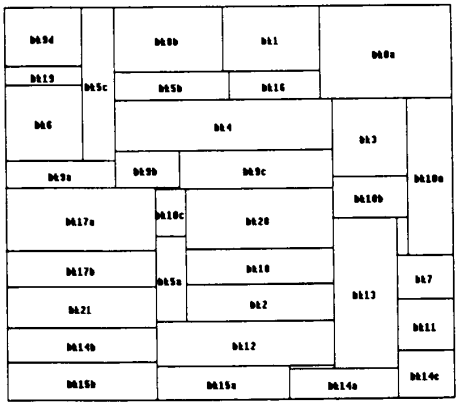


Figure 5. A floorplan of the aml33 chip.

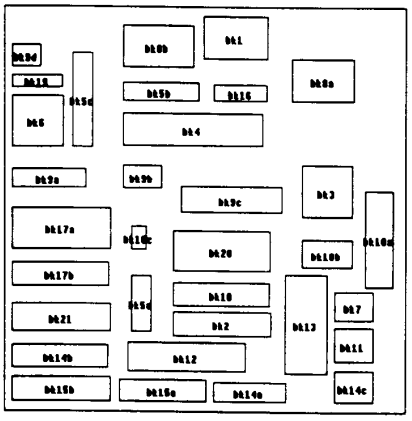


Figure 6. The final floorplan with routing space.