Timing Influenced General-Cell Genetic Floorplanner

Sadiq M. Sait Habib Youssef Shahid Tanvir M. S. T. Benten

Department of Computer Engineering
King Fahd University of Petroleum and Minerals
KFUPM # 673, Dhahran-31261, Saudi Arabia
e-mail: sadiq@ccse.kfupm.edu.sa

Abstract— In this paper we present a timing-influenced floorplanner for general cell IC design. The floorplanner works in two phases. In the first phase we restrict the modules to be rigid and the floorplan to be slicing. The second phase of floorplanner allows modification to the aspect ratios of individual modules to further reduce the area of the overall bounding box. The first phase is implemented using genetic algorithm while in the second phase we adopt a constraint graph based approach. Experimental results are also presented.

1 Introduction

The problem of timing sensitive floorplanning consists of determining suitable shapes and locations of cells, locations of pins on cells, locations of pads on layout, etc., while satisfying user specified timing constraints and minimizing certain objective functions.

For floorplanning, the aspects that need to be modeled consist of the components, the interconnections, the flexible interfaces (blocks and chip), the chip carrier (layout surface), any designer stated constraints, and the objective to optimize. A feasible floorplan optimizing the desired cost function is an optimum floorplan. Until recently, the size of the bounding box (comprising functional area and routing area) was a widely used measure of the quality of floorplan. However, due to advances in VLSI technology, sizes of transistors have been decreasing and their switching speeds increasing. This has increased the importance of interconnect delays with respect to the overall speed performance of the circuit. Hence, nowadays physical design steps such as placement and routing are made timing sensitive.

In this work we describe a timing sensitive floorplanner. Two reasons motivate making floorplanning sensitive to timing. (1) Since recently all subsequent stages of physical design such as placement and routing are made sensitive to timing, floorplanning stage must also be timing sensitive so that estimates of area, shapes, positions of pins on modules, etc., would be consistent with the placement objective. Therefore, for vertical consistency it is mandatory to incorporate timing information at the floorplanning stage. (2) The floorplanning stage when made sensitive to timing gives an estimation of the maximum clock rate of the circuit. This information can be used to tune the circuit early enough in the design process.

The floorplanning problem like other design automation problems is characterized by several noisy and conflicting objectives. This implies that constructive approaches will most likely miss reaching superior solutions. Iterative improvement approaches with hill climbing capability better explore the search space and offer a better chance of reaching the desired solutions. Genetic algorithm is suitable to such problems because of several reasons. (a) It is robust in that it consistently succeeds in locating a desirable solution from any random initial set of solutions; (b) it works on a population of solutions allowing a parallel search of the solution space; and (c) it is very convenient for problems with conflicting objectives such as floorplanning. However genetic algorithm is CPU time intensive and has large memory requirement. Two guidelines can be used to alleviate the above requirements, (1) choice of a suitable solution encoding and; (2) adoption of a simple model of the problem.

In this work we present a timing driven genetic floorplanner based on the above guidelines. The floorplanning problem is solved in two main phases. In the first phase, we adopt a simple floorplan model which has the following two restrictions: (a) solutions are restricted to slicing structures; and (b) all blocks are rigid but can have free orientations. The floorplan problem of the first phase is solved using a timing driven genetic approach. The second step is a floorplan resizing phase where blocks are allowed to have flexible shapes, and the floorplan solutions are no longer restricted to slicing structures. In this second phase modules maintain their topological proximity thus minimizing disturbance effects to the output of the first phase.

The novelty of this approach lies in using this two-step solution approach, where the solution from the iterative component phase is followed by a constructive refinement phase. This is in contrast to traditional iterative approaches which usually start from a constructively built solution which is improved using some generalized iterative improvement heuristic such as simulated annealing (SA) or genetic algorithm (GA). Further, our formulation is timing driven.

2 Literature Review

Various strategies have been reported in the literature for designing floorplans and can be broadly classified as constructive and iterative. Some of the constructive techniques use cluster growth, partitioning and slicing, connectivity clustering, and rectangular dualization. Other techniques are constraint graph based approaches, mathematical programming, or knowledge-based. Most of these techniques are discussed in detail in [6].

Iterative and non-deterministic techniques such as simulated annealing and genetic algorithms have also been employed to solve the floorplanning problem [3, 7]. There are two main variations of iterative algorithms as applied to floorplanning; namely direct and indirect. In the direct approach, manipulations are done directly on the physical layout of the floorplan. In the indirect approach, an abstract representation (such as a graph or slicing tree) of a floorplan is used for manipulations. The abstract solution is later mapped to obtain the physical representation of the floorplan. The work reported in [7] is an example of a direct approach using simulated annealing. The methods discussed in [10, 11] can be classified as an indirect approaches.

The issue of timing driven floorplanning has also been addressed. In [5], a procedure for path-delay constrained initial placement is presented which directly incorporates timing and geometrical constraints. The problem is modeled and mathematically formulated as a constrained non-linear programming problem. In [1], a floorplanning algorithm and a global router that uses a sequence of gradient descent operations based on force-directed functions are presented. The best floorplan is selected and overlaps are removed by applying simulated annealing. Circuit timing is also considered.

Much work has been done towards timing driven placement, whereas very little work has been done in the area of timing driven floorplanning. In this work we describe a timing influenced genetic floorplanner. The principle reason for selecting the genetic algorithm is that it allows several floorplan configurations to be maintained. This makes the technique easy to adapt to the multi-objective nature of the floorplan problem in general and timing-driven floorplanning in particular. Further, the chances of getting trapped in a local minima are reduced since several alternatives exist.

3 Timing prediction

The delay of the longest path in the circuit, which includes the delay due to both the logic cells and the interconnects, is not known prior to layout. And long path timing problems registered after layout are very difficult to correct because they may require, not only new iterations of the physical design steps, but possibly, many iterations of the logic design step.

Long path timing problems are caused by large interconnect delays. Obviously, the critical paths are those that are most inclined to exhibit a long path problem. In this work, the timing data passed by the timing analysis program to the floorplanning procedure consist of a set of the most critical paths. This set is predicted using the notion of α-criticality. Our prediction approach proceeds as follows. From past layouts with similar complexity, the average and standard deviation of net lengths are estimated for each type of net (2 pin-, 3 pin-,..., k pin-nets). These are converted to capacitances for the particular technology of the design at hand. Let T_π and S_π be the overall delay (including the net delay estimations) and standard deviation along path π, respectively. Let $T_{\max}$ be the estimated delay of the longest path in the design, that is,

$$T_{\max} = \max_{\pi}\{T_\pi\} \tag{1}$$

A path π is called *α-critical* if and only if,

$$T_\pi + \alpha \times S_\pi \geq T_{\max} \tag{2}$$

The parameter α acts as a confidence level. The larger α is, the larger is the number of predicted critical paths, and the higher is the probability of including all potentially critical paths. Typical values of α are: $\alpha \times S_\pi \leq 5ns$. This prediction approach was effective in predicting all of the critical paths in the designs we experimented with.

4 Timing Driven Genetic Floorplanning

In this section we explain the first phase of the floorplanner which is based on the genetic algorithm. Genetic algorithms (GA) are an effective optimization and search technique inspired by the mechanism of evolution and natural genetics [4]. They are characterized by a parallel search of the state space as against a point-by-point search by the conventional optimization techniques. The parallel search is achieved by keeping a set of possible solutions to the optimization problem, called *population*. An individual in the population is a string of symbols and is an abstract representation of the solution. The symbols are called *genes* and each string of genes is termed a *chromosome*. The individuals in the population are evaluated by some *fitness* measure. Based on the fitness value, two individuals (*parents*) are *selected* at a time from the population. The genetic operators (*crossover and mutation*) are applied on the selected parents to generate new possible solutions called *offsprings*.

The application of genetic algorithm for any problem requires a representation of the solution to the problem as a string of symbols, a choice of genetic operators, an evaluation function, a selection mechanism and determination of probabilities controlling the genetic operators. Each of these greatly influences the performance of the genetic algorithm.

Solution representation

The first phase of the floorplanner constrains the structure to be slicing and the circuit modules to be rigid, but allows rotation. The layout is represented as a repetitive division into basic rectangles by horizontal and vertical cut-lines. Such a layout is called a *slicing structure*. A slicing structure can be modeled by a binary tree with n leaves and $n-1$ nodes,

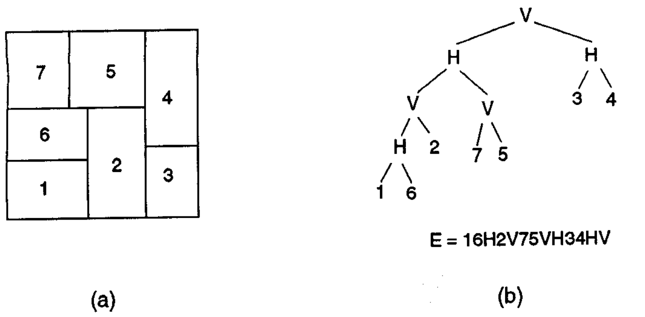

Figure 1: (a) A rectangular dissection. (b) Its corresponding slicing tree.

where each node represents a vertical cut-line or horizontal cut-line, and each leaf a basic rectangle. This binary tree is called a *slicing tree*, and the corresponding floorplan is a *slicing floorplan*. Letters H and V refer to horizontal and vertical cut operators respectively. A postorder traversal of a slicing tree produces a Polish expression with operators H and V, and with operands the basic rectangles $1, 2, \cdots, n$. Figure 1 gives a rectangular dissection, its corresponding slicing tree, and its Polish expression representation. Since there is only one way of performing a postorder traversal of a binary tree, there is one to one correspondence between floorplan trees and their corresponding Polish expressions. The operators H and V carry the following meanings: ijH means rectangle j on-top-of rectangle i; ijV means rectangle i to-the-left-of rectangle j. In this work each solution is encoded as a Polish expression [10].

Initial population generators

Two types of initial population generators were considered in our implementation, IPG_1 and IPG_2. IPG_1 constructs the Polish string by inserting $n-1$ operators in a random permutation of n operands. IPG_2 generates the Polish string in such a way that the modules are arranged in rows.

Fitness function

The fitness function consists of three terms representing the timing performance, the area of floorplan bounding rectangle, and the overall interconnection length. Since the above three quantities are incompatible, they are first normalized to a common mean and a common deviation. The fitness function of a solution i is equal to a weighted sum of three quantities as follows:

$$\text{Fitness}(i) = \frac{A^*(i)}{A_{\max}} \times W_A + \frac{T^*(i)}{T_{\max}} \times W_T + \frac{L^*(i)}{L_{\max}} \times W_L \quad (3)$$

where A^*, T^*, L^* represent normalized quantities of area, clock speed and wirelength respectively, for solution i. W_A, W_T, W_L are user specified relative weights for area, timing and wirelength respectively. The quantities, $A_{\max}$, $T_{\max}$, $L_{\max}$ are the maximum values in the given population. The quality of floorplan solution largely depends upon the values of the weights. By varying the weight values, the notion of optimality can be changed.

Selection for crossover

This step determines which parents are selected for crossover. In this work we used the selection mechanism based on the *proportionate selection* scheme implemented by the roulette wheel method [4]. In this selection scheme, each string is allocated a sector of a roulette wheel with the angle subtended by the sector at the center of the wheel which is equal to $2\pi \cdot \frac{f_i}{\sum_j f_j}$ where f_i is the fitness of solution i, and $\sum_j f_j$ is the sum of the fitness over the entire population.

Crossovers χ

Crossover is a mechanism for probabilistic inheritance of useful information from two individuals (parents) to offsprings. The main idea is that the genetic information of a good solution is spread over the entire population. Thus, the best solution can be obtained by thoroughly combining the chromosomes in the population. Four crossovers have been implemented and are invoked with different probabilities. The first three, namely χ_1, χ_2 and χ_3 were proposed in [3]. One more crossover, namely χ_4 is a variation of the partially mapped crossover (PMX) proposed for placement in [8].

Crossover χ_1, known as 'block inheritance crossover' generates a valid offspring by copying the operands from one parent (say P_1) insitu[1] and the operators from the other parent (say P_2) at the remaining positions. This crossover inherits the building blocks from one of the parents to the offspring.

Crossover χ_2, known as the 'slicing inheritance crossover', inherits the slicing structure from one of the parents. It is implemented by copying the operators from the first parent (say P_1) insitu and completing the offspring by copying the operands from the other parent (say P_2) at the remaining positions.

Crossover χ_3, the (sub-tree crossover) [2] is a mechanism for inheritance of a slicing sub-tree from the parent to the offspring. This is achieved by identifying a sub-tree in a parent slicing floorplan tree, and passing this sub-tree as is to the offspring. The operation of this crossover is explained as follows. Let P_1 and P_2 be the two parents to be crossed. First, the P_1 chromosome is scanned from left to right and a sub-tree is identified. The substring corresponding to the sub-tree is copied insitu in the offspring. Then the remaining operators are copied from the parent P_1 and the remaining operands are taken from the other parent P_2 to complete the offspring.

Crossover χ_4 implements the PMX crossover on the modules [8], and then, the operators are copied from each parent, thus resulting in two offsprings with the same order of modules, but different set of operators.

Mutation μ

Mutation is a means of introducing new information into the population. Our implementation uses five mutation operators μ_1 to μ_5. The first four are

[1]insitu means in the same location as that of the parent.

similar to the *perturbation* moves used by Wong and Liu in their simulated annealing solution to floorplanning [10] and are illustrated in Figure 2.

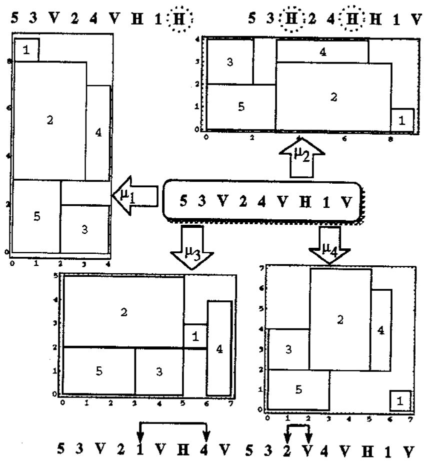

Figure 2: Illustrating the effect of mutation operators μ_1 to μ_4. μ_1: Complement an operator. μ_2: Complement a chain of adjacent operators. μ_3: Swap two adjacent operands. μ_4: Swap an operand with an adjacent operator.

A new mutation operator μ_5 called 'timing biased module exchange' is used. This operator pulls closer together those modules on a timing critical path with violations of the interconnect bounds. It works as follows: first, a net violating the net delay bound on a critical path is identified. Then, modules on this net are selected and brought closer together by swapping them with other modules not on critical paths.

Selection mechanism for next generation

A number of methods were proposed to select individuals that can survive and be used in the next generation [4]. In this work we use a technique that combines old population with the offsprings, selects the best individual, and then the remaining individuals are chosen probabilistically based on their fitnesses (higher fitness translates to higher survival probability).

In the next section, we describe the algorithm we use to refine the slicing floorplan produced by the genetic algorithm.

5 Floorplan Refinement

In this phase, we remove the slicing restriction on the floorplan structure. Also, modules are allowed to have flexible shapes. The refinement phase consists of two steps: (1) construction of a constraint set, and (2) shape optimization.

Graph construction

Two directed acyclic graphs are used to model the topological constraints between the blocks: a horizontal constraint graph G_H and a vertical constraint graph G_V. The vertex set of G_H is the set of blocks plus two dummy vertices: L, R. Similarly, the vertex set of G_V is the set of blocks plus two dummy vertices: T, B. The dummy vertices L, R, T, B correspond respectively to the left, right, top, and bottom boundaries of the layout. The edge set of G_H models the to-the-left/to-the-right relationships, while that of G_V models the on-the-top/on-the-bottom relationships.

A constraint set is *complete* if there exists a directed path between every pair of blocks b_i, b_j either in G_H, in G_V, or both. In a *strong complete* set each pair of blocks is *adjacent* either in G_H, in G_V, or both [9]. Two blocks are adjacent if they are connected by an edge. It is clear that a floorplan that satisfies a strongly complete set will have no overlaps. Obviously for two blocks not to overlap, only one constraint (either in the horizontal or vertical direction) is necessary and sufficient. Two blocks are called *overconstrained* if they are constrained in both the horizontal and vertical directions. The existence of overconstrained blocks negatively affects the area optimality of the floorplan. Our graph construction procedure is based on this key observation.

Definition 1 *A constraint set (G_H, G_V) is sufficiently constrained if there exists an edge between every pair of blocks (b_i, b_j) either in G_H or in G_V.*

G_H and G_V are constructed as follows. Two blocks are constrained if the center of one block must be to the left/below the center of the other. If two blocks (b_i, b_j) are overconstrained then the edge (i, j) is deleted from the graph G_H if the longest path traversing (i, j) in G_H is longer than the longest path traversing (i, j) in G_V. Otherwise the edge (i, j) is deleted from G_V. Therefore, the selection is based on which of the constraints will lead to a smaller-area floorplan. If two blocks are constrained in only one direction (i.e., they have the same x or y coordinate), the algorithm in this case has only one choice. This constructive process greedily generates a constraint set (i.e., G_H, G_V) according to Definition 1 and, at the same time, eliminates all redundant constraints right from the beginning. This is in contrast to the algorithm in [9], where only redundant edges belonging to the critical paths in G_H and G_V are examined. Removing all redundant constraints produces a more compact floorplan.

The next step in our floorplanner consists of reshaping the variable-shape blocks in order to optimize the floorplan area and satisfy the remaining constraints on block/chip aspect ratios.

Block reshaping

The reshaping algorithm determines dimensions and positions of the blocks so that floorplan area is minimized and constraints on block shapes are satisfied. This is achieved by iteratively reshaping flexible blocks on the longest paths.

The *reshaping* algorithm utilizes the constraint graphs G_H and G_V to compute the dimensions of the floorplan and decide on a candidate block for reshaping.

Suppose we want to reduce the floorplan dimension in the Y-direction without enlarging the floorplan in the X-direction. This can be achieved as follows. let $\ell(\pi_H)$ and $\ell(\pi_V)$ be the length of the *longest* paths in G_H and G_V respectively. Let block b_i be such that $b_i \in \pi_V$ and $b_i \notin \pi_H$. If π_H^i is the longest path traversing b_i in G_H, then the width w_i of b_i can be increased by an amount $\delta_i^x = \ell(\pi_H) - \ell(\pi_H^i)$ without increasing the overall area of the floorplan. δ_i^x is the maximum block's width increment that is guaranteed not to cause an increase in the length of the critical path π_H in G_H. But, since the block width has an upper bound w_i^{max}, then the legal δ_i^x is given by,

$$\delta_i^{x_{legal}} = \min(\delta_i^x,\, w_i^{max} - w_i) \tag{4}$$

Thus, the new dimensions w_i' and h_i' for block b_i are derived as follows,

$$w_i' = w_i + c \times \delta_i^{x_{legal}} \tag{5}$$

$$h_i' = a_i / w_i' \tag{6}$$

where c is a user specified positive real number ($0 < c \leq 1$) used to control how large the *x-increment* should be. Reshaping the blocks in small increments helps achieve a smaller floorplan with the correct aspect ratio. In our experiments, we set this parameter to 0.5. Optimization in the X-direction is similarly formulated.

The floorplan resizing process is terminated if there are no more blocks that can be selected for reshaping.

After completing the resizing process, the horizontal and vertical graphs are traced to determine the final xy-locations of the blocks. The lower left corner of the floorplan is at origin (0,0). The lower left corner of block b_i is placed at (x_i, y_i) where x_i is the longest L-to-b_i path in G_H and y_i is the longest B-to-b_i path in G_V.

Finally, the blocks are enclosed inside the smallest bounding rectangle with the desired aspect ratio ρ. The area of the bounding rectangle is the area of the floorplan.

6 Experiments and Results

There are several key parameters that affect the performance and behavior of GA. These are: (a) size of population, (b) initial population constructor, (c) selection mechanism for crossover, (d) type of crossovers and their probabilities, (e) mutation operators and their probabilities, (f) selection of individuals for next generation, and finally (g) cost function. These parameters are selected based on experimentation in the problem of interest. The size of the population depends on the size of the problem (number of blocks). It is recommended to use a large population (30) for small circuits (upto 30 blocks). For larger circuits a population size between 10 and 20 was used. Two types of initial population generators were considered, one constructs the Polish string by inserting $n-1$ operators in a random permutation of n operands and the other generates the Polish string in such a way that the modules are arranged in rows. The selection for crossover is as explained in Section 4. The current implementation applies all crossovers with the following probabilities: P_{χ_1}=.15, P_{χ_2}=.15, P_{χ_3}=.30, P_{χ_4}=.40, where P_χ represents the probability that crossover χ is applied on the selected two parents. Similarly, all mutation operators are also applied with the following probabilities: P_{μ_1}=.10, P_{μ_2}=.05, P_{μ_3}=.10, P_{μ_4}=.40, P_{μ_5}=.10.

We experimented with test cases of sizes varying between 20 and 125 modules. For each circuit the floorplanner is supplied with a set of the predicted most critical paths. As explained in Section 4, the fitness function is a weighted sum of the area of the floorplan bounding rectangle, the wirelength of the interconnects, and the circuit clock period. We experimented with several weight assignments.

In Table 1 we report the results corresponding to three different weight assignments. In the first column, 'Area Only', the fitness includes only the area term (wirelength and clock period are given zero weights). In the second column, 'Area+Wire', the fitness includes the area and wirelength term with equal weights (50% each). It may be observed that for all test cases the wirelength has consistently improved. A decrease in wirelength between 5% and 14% was achieved with no increase in area.

In the third column, 'Area+Wire+Time', the fitness includes all terms with weights 50%, 25% and 25% for area, wirelength, clock period, respectively. Note that the lower bound on the clock period is given by the maximum path delay due to logic only (column Max Delay in Table 1). In this third case we observed a decrease in the interconnect delay between 20% and 40%. For example, for the Parity3 circuit, the clock period when area was the only objective was 46.4 nano seconds. Since the logic delay on the path is 27.134 nano seconds, the delay due to interconnects is 19.266 nano seconds. When timing was included in the fitness function, the delay due to interconnects was reduced to 12.306 nano seconds. Thus a reduction in interconnect delays by 36% was achieved. The increase in area of the bounding rectangle in this third case when all terms are weighted is upto a maximum of 12%. Note that the dead space introduced in this stage is further reduced by the second phase of the floorplanner.

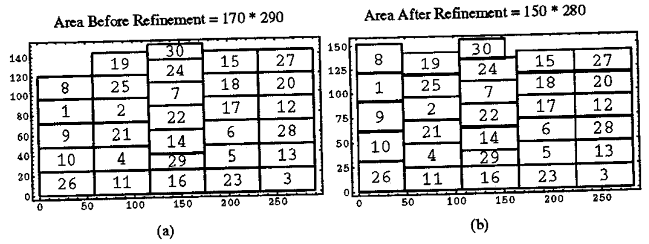
Area Before Refinement = 170 * 290
Area After Refinement = 150 * 280
(a)
(b)

Figure 3: (a) Slicing floorplan obtained from phase-1; (b) floorplan obtained after phase-2.

Figure 3 illustrates the effect of the floorplan refinement phase on the Parity3 circuit. The relative positions of the blocks are retained but the area of the bounding box has been reduced further by about 15%.

Test	No. of	Max	Area Only			Area + Wire			Area+Wire+Time		
Circuits	blocks	Delay	Area	Wire	Clock	Area	Wire	Clock	Area	Wire	Clock
Parity2	20	36.830	42320	5051	45.87	41088	4579	44.24	47580	4091	42.0
Parity3	30	27.134	43616	7573	46.40	43616	6514	43.37	43616	6124	39.44
Highway	45	15.540	104690	14527	29.35	102200	13805	31.25	108016	13013	25.20
Fract	125	34.14	322392	82299	86.28	319680	76100	78.2	319680	76140	74.4

Table 1: Results of experiments considering different objectives. Column 'Max Delay' contains the delay due to logic elements only. Column 'Clock' gives the delay of the longest path which includes logic and interconnect delays.

In these experiments the interconnect delays were inflated by a factor of 3 in order to make the timing aspects more pronounced. Current implementation does not take into account routability or pin location issues. The reason is that routability is usually not a problem for the general cell design style and can be easily incorporated into our system by integrating a global router with the floorplan sizing procedure of the second phase. The task of the global router is to compute routing space requirements between blocks. These space requirements can be specified as edge weights in the constraint graphs G_H and G_V. Then, the algorithm will use the edge and node weights to compute correct locations of all the blocks as well as the final dimensions of the floorplan.

7 Conclusions

In this paper we presented a novel approach to timing-driven floorplanning. The novelty of the presented approach comes from using a simpler representation in the first phase to solve a difficult problem, and then using a constructive technique in the second phase to refine the solution while easing the restrictions of the first phase. Timing constraints are included in the cost function.

The floorplanning procedure follows the genetic algorithm. The program uses net and path timing data from a timing analyzer. The timing analyzer uses the notion of *α-criticality* to predict the critical paths prior to floorplanning. Extensive experiments were conducted to tune the parameters of the program and evaluate the extent of timing improvement.

Acknowledgment

Authors acknowledge King Fahd University of Petroleum and Minerals for support under KFUPM Project # COE/VLSIDESIGN/162. We also acknowledge K. Al-Farrah for implementing the timing predictor and the reshaping algorithms.

References

[1] D. R. Brasen and M. L. Bushnell. MHERTZ: A New Optimization Algorithm for Floorplanning and Global Routing. In *Proceedings of the 27th ACM/IEEE Design Automation Conference*, pages 107–110, 1990.

[2] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Punctuated equilibria: A parallel genetic algorithm. In *Proceedings of the Second International Conference on Genetic Algorithms*, pages 148–154, 1987.

[3] J. P. Cohoon, S. U. Hedge, W. N. Martin, and D. Richards. Distributed genetic algorithms for the floorplan design problem. *IEEE Transactions on Computer-Aided Design*, 10(4):483–492, 1991.

[4] D. E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning.* Addison-Wesley Publishing Company, INC., 1989.

[5] S. Prasitjutrakul and William J. Kubitz. Path-Delay Constrained Floorplanning: A Mathematical Programming Approach for Initial Placement. In *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pages 364–369, 1989.

[6] Sadiq M. Sait and Habib Youssef. *VLSI Physical Design Automation: Theory and Practice.* McGraw-Hill Book Co., Europe (also co-published by IEEE Press), 1995.

[7] C. Sechen. Chip Planning, Placement, and Global Routing of Macro/Custom Cell Integrated Circuits Using Simulated Annealing. In *Proceedings of the 25th Design Automation Conference*, pages 73–80, June 1988.

[8] K. Shahookar and P. Mazumder. VLSI Cell Placement. *ACM Computing Surveys*, 23(2):143–220, June 1991.

[9] G. Vijayan and R. Tsay. A new method for floor planning using topological constraint reduction. *IEEE Transactions on CAD*, 10(12):1494–1501, December 1991.

[10] D. F. Wong and C. L. Liu. A new algorithm for floorplan design. *Proc. of the 23rd DAC*, pages 101–107, 1986.

[11] D. F. Wong and Khe-Sing The. An algorithm for hierarchical floorplan design. *Proc. of the ICCAD*, pages 484–487, 1989.