RITUAL
AN ALGORITHM FOR PERFORMANCE-DRIVEN PLACEMENT OF CELL-BASED ICs

by

A. Srinivasan, K. Chaudhary, and E.S. Kuh

Memorandum No. UCB/ERL M91/47

28 May 1991

RITUAL
AN ALGORITHM FOR PERFORMANCE-DRIVEN PLACEMENT OF CELL-BASED ICs

by

A. Srinivasan, K. Chaudhary, and E.S. Kuh

Memorandum No. UCB/ERL M91/47

28 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

RITUAL
AN ALGORITHM FOR PERFORMANCE-DRIVEN PLACEMENT OF CELL-BASED ICs

by

A. Srinivasan, K. Chaudhary, and E.S. Kuh

Memorandum No. UCB/ERL M91/47

28 May 1991

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

RITUAL

An Algorithm for Performance-Driven Placement of Cell-Based ICs

Abstract

In this paper we describe an efficient algorithm for obtaining a placement of cell-based ICs subject to performance constraints. Using sophisticated mathematical techniques, we are able to solve large problems quickly and effectively. The algorithm is very simple and elegant, making it easy to implement. In addition, it yields good results as we show on a set of real examples. On the average, we are able to make 20% improvement in the wire delay of these examples with little or no impact on the total Steiner tree wirelength. The acronym RITUAL represents the key idea of our technique: Residual Iterative Technique for Updating All Lagrange multipliers.

1 Introduction

As interconnect wires on ICs are scaled to smaller dimensions, the performance of chips becomes dominated by wire delay. Thus physical design tools of today need to address performance issues at every stage of the design hierarchy. In this paper, we focus on performance-oriented placement. Performance during the placement phase of physical design has been considered by many researchers in the past and can be broadly grouped into two categories: net-based and path-based approaches. Physical design is a net-based process, i.e., the physical design tools operate on nets and the objects to which they connect. Timing constraints are inherently path-based, and such constraints place requirements on the total delay of a well defined sequences of modules and nets with.

Net-based approaches are discussed in [1]-[7],[15],[16]. In some net-based algorithms, weights are assigned to nets to reflect the criticality of paths which may be determined by a timing verifier statically or dynamically. In other net-based approaches, a pre-timing analysis may be used to derive maximum bounds on the sizes of the nets. The problem with net-based approaches is that there is no guarantee that fixing bounds for a set of nets will not make another set of paths critical and individual net bounds may be overconstraining.

In path-based approaches like [11], the path nature of timing constraints and the physical representation of the IC are unified in a single formulation. However the problem with the approach of [11] is that it takes too long even on moderate-sized examples. The authors of [14] use a constructive method of placing gates sequentially, with a cost function that tries to capture timing behavior, but cannot guarantee satisfaction of the timing model. In [16], the authors defines regions for critical paths in which all the modules on the paths must be placed, and then use constructive placement. This method also can not guarantee the satisfaction of timing constraints and suffers from the sequential nature of constructive placement.

Unlike previous approaches that used heuristic methods like net-weighting or placing bounds on net lengths, we dynamically model timing behavior of all paths efficiently during placement thus removing the need for heuristics. Our timing model is based on net capacitance which is the "major performance limiter" according to [8].

2 The Mathematical Models

An IC may be viewed as a collection of modules (or cells) interconnected by nets that attach to the modules at pins (or terminals). Let

$$\mathcal{M}=\{ m_1, ..., m_M\}, \mathcal{N}=\{ n_1, ..., n_N \} \text{ and } \mathcal{P}= \{p_1, ..., p_P\},$$

respectively denote the sets of modules, nets, and pins. The modules can be categorized by function as: combinational, synchronizing, primary input (PI), and primary output (PO). Let f represent the number of primary inputs, and g represent the number of primary outputs; thus, there are M-f-g internal modules where an internal module is defined to be inside the periphery of the chip with freedom to move.

2.1 Wirelength Model

Let x_{pi} and y_{pi} denote the x and y coordinates of pin p_i on the chip. We use the following estimator for the length of net n.

The estimate L_n is the square of the Euclidean distance between the pins on the net *n*. For the type of

$$L_n = \sum_{pi, pj \in \mathcal{N}} ((x_{pi} - x_{pj})^2 + (y_{pi} - y_{pj})^2)$$

ICs we consider (small-cell ASICs), this estimate has been shown to be accurate by [14] and references therein and has been widely used in practice. It is assumed that the pins of a module are located at the center of the module and the module's location is represented by a single (x, y) coordinate that coincides with the center of the module on the chip.

2.2 Total Wirelength

With the assumption that the pins on a module and the module share the same location, the expression for the estimate of the cost of a placement is

$$L = 1/2 \sum C_{ij}((x_i - x_j)^2 + (y_i - y_j)^2) \quad (1)$$

where C_{ij} represents the number of nets that modules m_i and m_j share. (x_i, y_i) and (x_j, y_j) represent the locations of m_i and m_j.

The modules are partitioned into two sets, fixed and movable. Fixed modules are IO pads or modules that have been assigned a location on the chip, for example, clock pads. Movable modules have variable x and y coordinates. The cost function can then be rewritten using matrix notation as

$$L(x,y) = 1/2(\mathbf{x}^T\mathbf{B}\mathbf{x} + \mathbf{y}^T\mathbf{B}\mathbf{y}) + \mathbf{c}^T\mathbf{x} + \mathbf{d}^T) \quad (2)$$

where x is a vector of the x-coordinates of the module locations and y is a vector of the y-coordinates. **c** and **d** are contributions from fixed modules. **B** is a symmetric matrix with

$$\mathbf{B} = \mathbf{D} - \mathbf{C} \quad (3)$$

where $\mathbf{C} = [c_{ij}]$ and **D** is a diagonal matrix with $d_{ii} = \Sigma_{j=1,n} c_{ij}$. If the modules cannot be partitioned into disconnected subsets, then **B** is positive semi-definite (see [14]). In addition, **B** is almost always sparse

for practical gate-array and sea-of-gate circuits. This enables efficient numerical techniques to be applied to the matrix.

3 Timing Model

In this paper synchronous performance optimization is addressed. The long path problem is considered and the related short path problem is ignored. The work could be extended to deal with both problems. For simplicity it is assumed that edge-triggered synchronizing elements are used. The methods described are generalizable to the case of level-sensitive latches.

Let a digraph $D_T(V,A)$ represent the integrated circuit in the timing domain. Let the vertex set V be in one-to-one correspondence with the pins. Arc weights $d(v_i,v_j)\ \forall\ (v_i,v_j) \in A$ denote the pin-to-pin signal propagation delays, and arc directedness represents the direction of signal flow in the circuit. Also, let $A = A^I \cup A^E$ respectively model the signal behavior *internal* and *external* to all cells; thus, internal signal arcs represent cell signal flow while external arcs represent net signal flow. A path ψ, is defined by the sequence $(v_s, \ldots, v_e)$ of vertices that lie on the path. The delay of module m_i is characterized by $d(v_i,v_j)\ \forall\ (v_i,v_j) \in A^I$ and of net n_i is characterized by $d(v_i,v_j)\ \forall\ (v_i,v_j) \in A^E$. This multiple-arc cell and net model permits more accurate modeling than single cell and net delay models due to its greater flexibility. Associated with each path endpoint vertex is a required arrival time r_i. Associated with each path starting point vertex is an actual arrival time a_i. The worst-case actual arrival time a_j is given by

$$a_j = \max \{a_i + d(v_i,v_j) \mid \forall\ (v_i,v_j) \in A\} \tag{4}$$

The required arrival time ri is defined to be

$$r_i = \min \{r_j - d(v_i,v_j) \mid \forall\ (v_i,v_j) \in A\} \tag{5}$$

The quantities may be computed by a breadth-first search. Based on the calculation of actual arrival and required arrival times for all v_i, a slack s_i may be defined as $s_i = r_i - a_i$. A negative value of s_i for v_i indicates that a violation of a timing constraint has occurred.

Definition *The timing of the chip is said to be feasible if and only if* $s_i \geq 0, \forall\ v_i \in V$.

3.1 The Long Path Timing Problem

The longest path delay through combinational logic corresponds to the earliest time at which the outputs settle. The relationship between the longest path delay T_{long}, the clock period CP, the skew to the synchronizing clock pins T_{skew}, the set-up time of the synchronizing elements T_{su}, and the internal clock to output delay $T_{clk \rightarrow Q}$ of synchronizing elements is:

$$CP \geq T_{long} + T_{skew} + T_{clk \rightarrow Q} + T_{su} \tag{6}$$

If equation 6 is not satisfied, then a long path timing problem exists in the design. A long path is defined as follows:

Definition A critical long path Π is a path Ψ in which the sequence of vertices $(v_s, \ldots, v_e)$, $v_s \in S$ and $v_e \in E$ comprising the path all have slack values less than or equal to zero.

$$\Pi = \{v_i \mid s_i \leq 0\ \forall\ v_i \in \psi\}$$

Thus, a necessary and sufficient condition for the existence of no long path problem is $s_i > 0, \forall v_i \in V$.

4 Problem Formulation

This section describes a formulation that uses first-order delay functions for the arcs in D_T and an efficient representation for the resulting delay equations. Although the algorithm is easily generalizable to arbitrary convex delay functions, the linear delay model is used for simplicity.

Let C_h and C_v denote the horizontal and vertical capacitance per unit length of the horizontal and vertical interconnect wires respectively. Let R_i denote the output resistance of module m_i. Let m_j be a fanout of m_i. The wire delay between m_i and m_j (and the arc weight $d(v_i, v_j)$, $(v_i, v_j) \in A^E$ is determined by the following equation

$$d(v_i, v_j) = R_i [C_h |x_i - x_j| + C_v |y_i - y_j|] \quad (7)$$

Although this delay equation is non-linear, a mathematical device can be used to convert it to an equivalent linear equation. This technique is described in a subsequent section. To simplify further discussion, the following notation is introduced. Let

$$\mathbf{w}_{cell} = \begin{bmatrix} \mathbf{x} \\ \mathbf{y} \end{bmatrix}$$

be the combined vector of x and y coordinates of cell positions. Let $\mathbf{w}_{time}$ denote the vertex actual arrival time variables. Then

$$\mathbf{w} = \begin{bmatrix} \mathbf{w}_{cell} \\ \mathbf{w}_{time} \end{bmatrix}$$

is the 2M + P vector of all variables in the formulation of the problem. However, the extra (P) variables corresponding to the arrival times do not enter into the cost function, so the value of the cost function at any point is unchanged and the sparsity of the matrix representing the cost function is retained. Let

$$Q = \begin{bmatrix} B & 0 & B \\ 0 & B & 0 \\ 0 & 0 & B \end{bmatrix}$$

be the combined (2M+P)×(2M+P) matrix for the cost function and let

$$b = \begin{bmatrix} c \\ d \\ 0 \end{bmatrix}$$

Then the cost function can be writen as

$$L = 1/2(\mathbf{w}^T Q \mathbf{w} + \mathbf{b}^T \mathbf{w}) \quad (8)$$

The problem of minimizing wirelength subject to timing constraints be stated as:

minimize L	(NLP)
subject to	
$a_j \geq a_i + d(v_i, v_j)$	$\forall (v_i, v_j) \in A$
$a_j \leq T_e$	$\forall v_j \in E$

$$a_j \geq T_s \qquad \forall v_j \in S \tag{9}$$

where $d(v_i, v_j)$ is given by Equation 7.

Theorem 1 : If there exists at least one fixed module and the modules do not form disconnected subsets, then $\mathbf{x}^T\mathbf{B}\mathbf{x}$and $\mathbf{y}^T\mathbf{B}\mathbf{y}$ are positive definite.

Corollary 1.1 : Any relative minimum of NLP is also a global minimum.

Corollary 1.2 : The satisfaction of the Kuhn-Tucker first-order optimality conditions are sufficient for a point to be a global minimizer of NLP.

Active constraints at a point are defined to be those constraints that are satisfied with equality. Let A^* denote the vector function (possibly non-linear) of the active constraints at a global minimum w^* and ∇A^* denote the associated Jacobian matrix. It is assumed that ∇A is well-defined in the feasible region of the constraints. Corollary 1.2 states that there exist Lagrange multipliers λ satisfying

$$\begin{aligned} \nabla L(w^*) + \lambda \nabla A^* &= 0 \\ \lambda \nabla A^* &= 0 \\ \lambda &\geq 0 \end{aligned} \tag{10}$$

provided w^* is a regular point of the constraints, i.e., at w^* the matrix ∇A^* has full rank. In case the constraints are linear, $\nabla A^* = A^*$ and the equations are:

$$\begin{aligned} \nabla L(w^*) + \lambda A^* &= 0 \\ \lambda A^* &= 0 \\ \lambda &\geq 0 \end{aligned} \tag{11}$$

5 Resolving Slot Constraints

A common requirement in most cell-based ICs is that the cells lie in slots or regular arrays. A solution of NLP will yield a "placement" that usually does not satisfy the slot requirements. Such a placement has been called an "initial" or "global" placement. Several techniques have been proposed to refine the global placement to produce a slotted final result [15], [14]. The technique that we use is generalization of that proposed in [15]. The key feature of our technique is the highly efficient method we use to solve the problem with slot resolution constraints in the presence of timing constraints. The solution technique will be presented in a later section.

Let r^x and r^y represent the coordinates of the center of the chip. We first solve the global timing-constrained placement problem with two additional constraints:

$$1/M \sum_{i \in \mathcal{M}} x_i = r^x$$

$$1/M \sum_{i \in \mathcal{M}} y_i = r^y$$

These constraints ensure that the cells are spread around the center of the chip. After this, we partition the cells into four equal sized sets. This is done by first dividing the cells into two equal sized sets along the y-direction and then subdividing each set into two subsets along the x-direction. Let the sets be S_0, S_1, S_2 and S_3. We also divide the chip into four equal-sized regions and (r_i^x, r_i^y) denotes the coordinate of the center of the ith region. Figure 1 shows an example with four regions and the sets of cells in different shades after the solution. (Note that some cells from one region have migrated into

another). Now, we add eight centering constraints to the constraint set to form a new problem NLP1.

$$1/|S_i| \sum_{i \in S_i} x_i = r^x, \qquad i = 1, ..., 4$$

$$1/|S_i| \sum_{i \in S_i} y_i = r^y, \qquad i = 1, ..., 4$$

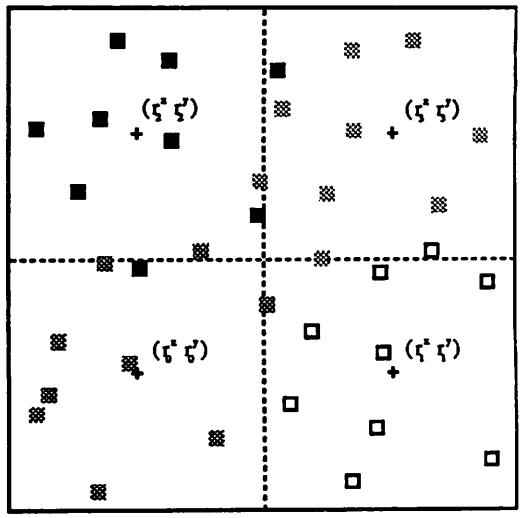


Figure 1: Cells after solution with eight partitioning constraints

The effect of these constraints is to spread the cells out in the four directions. Note that unlike many other partitioning approaches, a cell is not required to lie within its region. A cell has freedom to migrate into any other region. This allows the algorithm greater flexibility in minimizing wirelength while still satisfying slot constraints. It also makes the partitioning of cells more flexible in that a cell may change partitions later in order to reduce wirelength or satisfy timing constraints. Following the solution of NLP1, we perform repartitioning of the cells into sixteen sets, giving thirty two center of mass equations, and we the solve the new problem NLP2 (with the timing constraints included). During the repartitioning, the old partition information is not considered and new partitions are generated based on current cell locations. The solution and repartitioning process can be repeated to a level of granularity such that one cell remains within each region. This technique effectively resolves the slotted array requirements of cell-based ICs.

6 Why do standard methods fail?

The number of constraints in NLP can be enormous even for problems of moderate size. There are at least four active bounding-box constraints for each net and the Lagrange multipliers for those constraints not on critical paths have zero value. Similarly, for every pin, there is one active timing constraint - the one that corresponds to the maximum in equation 8. Again, the multipliers for the timing constraints not on critical paths are zero. The number of active variables in conventional techniques can be enormous. For a typical problem with 1000 cells and 3000 nets, the number of active variables could be upto 18,000 and the active constraints could number 15,000. However, what makes the problem even more difficult is that the constraint set is highly degenerate (see [9]). The effect of degeneracy is that standard quadratic-programming algorithms flounder for many steps without improving the objective function.

7 An Outline of the RITUAL

The key feature of our work is the method we use to solve the large-scale optimization problem in a very efficient manner. An interesting feature of our approach is that the number of variables and need to solve for any time is $2 \times M$ where M is the number of cells. In addition we do not encounter the degeneracy problem. We use a Lagrangian Relaxation method to solve the non-linear programming problem (see [17] and further references therein) at every level of partitioning. By developing specific

techniques that take advantage of the structure of the timing and slot constraints and objective function of NLP, we are able to solve the problem very efficiently.

Some of the features of the method are:

- Memory requirements are linear in the size of the problem
- An iterative technique that is very fast
- The problem can be solved to any desired accuracy
- Generalizable to arbitrary convex delay functions
- All critical paths are considered in a very efficient manner
- Slot resolution constraints are integrated in an efficient and consistent manner with the timing constraints
- The problem is solved optimally at every level of partitioning

In this section we describe the general outline for solving NLP by residual iterative update of Lagrange multipliers. The technique is specially suited to problems which are very easy to solve (or have special structure) if the constraints or a subset of constraints is removed. For simplicity, we assume that the constraint set is linear and the objective function is convex. Let the optimization problem be:

$$\text{minimize } f(\mathbf{x}) \qquad \text{subject to } \mathbf{Ax} \leq \mathbf{b} \tag{12}$$

Let **A** have m constraints. A Lagrangian problem is created from this by defining an m vector of Lagrange multipliers λ and adding the term $\lambda(\mathbf{Ax}-b)$ to the objective function to obtain the following Lagragian function

$$\max_{\lambda} \min_{x} f(\mathbf{x}) + \lambda(Ax-b) \tag{13}$$

The general flow of the algorithm is:

1. Select some initial value for λ.
2. Solve Equation 13. This is easy since for fixed λ the problem reduces to an unconstrained one (or if **A** is subset of the constraints, then it is easy to solve with the remaining simpler constraints).
3. Update λ
4. Check for termination conditions and if the problem is not solved, go to step 2

In general, this recipe needs to be adapted to specific problems and steps step 2 and 3 require careful consideration for different classes of problems. We have developed special techniques for step 2 and 3 which enable us to apply the algorithm successfully to the performance-driven placement problem.

7.1 Solving the Lagrangian

For the performance driven placement problem, the Lagrangian can be represented as:

$$\text{minimize } 1/2\, \mathbf{w}^T\mathbf{Q}\mathbf{w} + \lambda(\mathbf{Aw}-\mathbf{b}) \tag{14}$$

where **A** is the matrix of timing and center of mass constraints. For any fixed value of λ, say λ^k *the* problem has a very simple solution.

$$\mathbf{w}^{k+1} = \mathbf{Q}^{-1}\,[\lambda^k(\mathbf{A}\mathbf{w}^k-\mathbf{b})] \tag{15}$$

Note that Q is independent of cell locations. Thus, at every iteration, only the right-hand side of Equation 15 changes. Q^{-1} can be maintained in factored form (Cholesky factors are an excellent choice, [9]). Most of the work at every iteration involves only "forward" and "backward" substitution which can be done extremely quickly for sparse factors. This is one of reasons for the substantial speed of our algorithm.

7.2 Updating the Lagrange Multipliers

The method we use to update Lagrange multipliers from iteration to iteration is based on the subgradient method for setting dual variables [17]. This technique starts with an initial value λ^0 and iteratively applies the formula:

$$\lambda^{k+1} = \max \{0, \lambda^k - t_k(\mathbf{A}\mathbf{w}^k - \mathbf{b})\} \quad (16)$$

In this formula, t_k is a scalar step size and $\mathbf{w}^k$ is the optimal solution for Equation 13 for $\lambda = \lambda^k$. The choice of t_k is critical to the success of the algorithm because of the absolute valued delay constraints and the procedure for computing them is explained in a following subsection. The convergence properties of such a method are described in detail in [17].

7.3 Computing t_k

Suppose we are currently at a solution $\mathbf{w}^k$. At this solution, for all the critical paths, we write the delay as linear equations, removing the absolute values from Equation 7, switching signs wherever necessary to ensure that all the terms are non-negative. For example if currently $x_2 > x_3$ and there is a critical path passing from cell 3 to cell 2, we write $x_2 - x_3$, otherwise, we write $x_3 - x_2$. Then, we update the right hand side of Equation 15 and solve for $\mathbf{w}^{k+1}$. Now, we select the largest value of t_k such that a term in one of the delay equations just changes sign. We update $\mathbf{w}^{k+1}$ and λ^{k+1} according to this value of t_k.

7.4 Updating critical path set

After solving for $\mathbf{w}^{k+1}$, we perform a fast timing analysis on the timing graph to determine the paths that have become critical since the previous iteration and add them to the critical set with zero Lagrange multipliers. Note that because the timing equations (Equations (4), (5)) are stated in terms of the worst case path passing through a cell, we need to add only the worst path through the cell. This ensures that only a linear (in the size of the timing graph) number of equations ever need to be added even though the actual number of critical paths may be very large.

8 The RITUAL

The algorithm is described in Figure 2. The work done per inner-loop iteration of the algorithm is very little since it involves a right-hand side update which is O(M), one step of forward and backward substitution to solve for the new value of $\mathbf{w}$ which is $O(M^2)$, computing t, which is O(E), where E is the number of edges in D_T, and updating the critical paths, which can be done in O(M+E). Therefore, the work done per inner-loop iteration is $O(M^2)$. Note that the critical path set is continuously updated as new paths become critical. For the linearized delay equations, this procedure converges [17]. There is no theoretical bound on the number of iterations required for convergence of the inner loop, however, we found that in practice, the number of iterations required per level was very low - 200-400.

```
begin RITUAL
        for level = 0 to n_levels
                partition cells into regions;
                add partitioning constraints to constraint set;
                initialize Lagrange multipliers;
                while ( max error in active constraints  > ε )
                        update rhs of Equation 15;
                        solve for w;
                        compute t;
                        update w and Lagrange multipliers;
                        update critical path set and signs of delay terms;
                endwhile;
        endfor;
        assign cells to the slots using slot_assign();
end RITUAL;
```

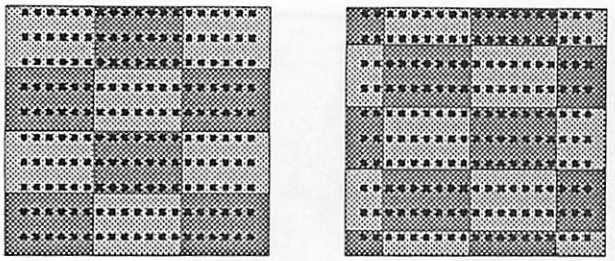
Figure 2: Flow of the algorithm

9 Resolving Slot Constraints in Practice

Although it is possible to add slot constraints as described in Section 4 till exactly one cell remains in each region, we found that using a modification of that technique results in further improvement in timing and wirelength.

We perform hierarchical partitioning until 10-20 cells remain in each region. Following this, we use Linear Assignment (weighted bipartite matching) [20] to assign slot positions to the cells within each region. We generate a cost matrix C with rows equal to the number of the cells in the region and as many columns as the number of slots in the region. An entry C_{ij} in the cost matrix is the cost of assigning i^{th} cell to the j^{th} slot. For non-critical cells the cost is the sum of the Manhattan wire length to the fanins and fanouts of the cell, where as for critical cells we only consider the critical fanins and fanouts. Furthermore, the wire length for the critical cells is multiplied by the driving resistance of the cells to make the cost propotional to the path slacks, and a user defined weigthing factor β to adjust the relative importance of timing with respect to the wire length.The Linear Assignment assigns a slot to each cell in such a way as to miminize the total cost .

Note that the solution is locally optimal for a region only with respect to the connections outside the region, and does not guarantee to minimze the cost of connections within the region. The problem could have been formulated as a Quadratic Assignment to handle the connections within the region properly. However, the large run time for Quadratic Assignment makes it impractical for regions with large number of cells. For regions with fewer cells, the effect of interconnection within the region is small, and by repeating the Linear Assignment few times an improved solution can be obtained. A further improvement in wirelength and timing can be obtained by allowing cells to migrate outside their region. This is achieved by shifting the regions in x and y directions by half the region size at alternate iterations as shown in Figure 4 and repeating the process until the improvement is small. The



(a) *Regions for linear assignment during even iterations.*

(b) *Regions for linear assignment during odd iterations.*

Figure 4

flow of the slot assignment algorithm is shown in Figure 5. Although slot assignment could be applied to any solution, we observed that following RITUAL by slot assignment yielded significantly better results than applying slot assignment directly.

10 Input/Output Assignment

Significant improvements in delay and wirelength can be obtained by reassigning the input and output locations on the boundary of the chip. We perform this assignment by using Linear Assignment technique described in the previous section to the inputs and output cells. After reassigning the inputs and outputs, we repeat the entire optimization process. The flow of the complete performance-driven placement algorithm, SPIRITUAL is shown in Figure 6.

```
begin SLOT_ASSIGN
        partition cells into regions based on RITUAL result;
        while ( improvement in wirelength )
                for each region
                        construct cost matrix;
                        perform linear assignment;
                        update cell locations;
                endfor;
                shift regions on chip;
        end while;
end SLOT_ASSIGN;
```

Figure 5: Flow of slot assignment algorithm

```
begin SPIRITUAL
        for user specified number of times
                RITUAL;
                IO_ASSIGN;
        end for;
end SPIRITUAL;
```

Figure 6: Flow of complete algorithm

Example	cells	RITUAL				Gordian	
		Without timing		*With Timing*			
		delay	*wl*	*delay*	*wl*	*delay*	*wl*
C1908	629	37.6	344	34.4	426	38.8	345
C1355	659	38.5	330	35.8	352	41.6	355
C2670	826	42.9	740	34.8	788	45.8	756
C3540	1137	62.7	919	51.1	1085	59.7	943
C5315	1337	38.7	1025	32.5	1135	43.0	1117
C7552	2253	45.7	1623	37.4	1774	47.4	1821
des	2591	51.2	2902	41.6	2989	47.9	3005
s5378	1377	27.3	1462	21.6	1679	27.3	1426
s15850	4446	78.7	5473	58.8	5563	86.6	5445

Table 1

Example	cells	RITUAL				Gordian	
		Without timing		*With Timing*			
		delay	*wl*	*delay*	*wl*	*delay*	*wl*
C1908	629	1.0	1.0	0.91	1.24	1.03	1.00
C1355	659	1.0	1.0	0.93	1.07	1.08	1.08
C2670	826	1.0	1.0	0.81	1.06	1.07	1.02
C3540	1137	1.0	1.0	0.81	1.18	0.95	1.02
C5315	1337	1.0	1.0	0.84	1.10	1.11	1.09
C7552	2253	1.0	1.0	0.82	1.09	1.04	1.12
des	2591	1.0	1.0	0.81	1.03	0.94	1.04
s5378	1377	1.0	1.0	0.79	1.15	1.00	0.98
s15850	4446	1.0	1.0	0.75	1.02	1.10	0.99
Average		**1.0**	**1.0**	**0.83**	**1.10**	**1.04**	**1.04**

Table 2

11 Results

We implemented RITUAL in C on a DECStation 3100. The algorithm was tested on an ISCAS set of benchmark examples using parameters for 1 micron CMOS technology. The examples were generated by the logic synthesis system MISII [21]. The run time even for the largest example with 4446 cells was less than 15 minutes of CPU time. The examples were partitioned to a level of granularity such that 15-25 cells were left in each partition followed by the slot assignment algorithm (IO pad assignment was not used for these results). We ran RITUAL with timing and without timing optimization and compared the result with an industry standard placement package Gordian[19]. Gordian uses the quadratic wire length and slicing optimization and was developed at Siemens Inc. In every case the timing was improved, and surprisingly, the wire length was also reduced for all large examples (see Table 2.) The wirelength measure we used for comparison is the single-trunk steiner

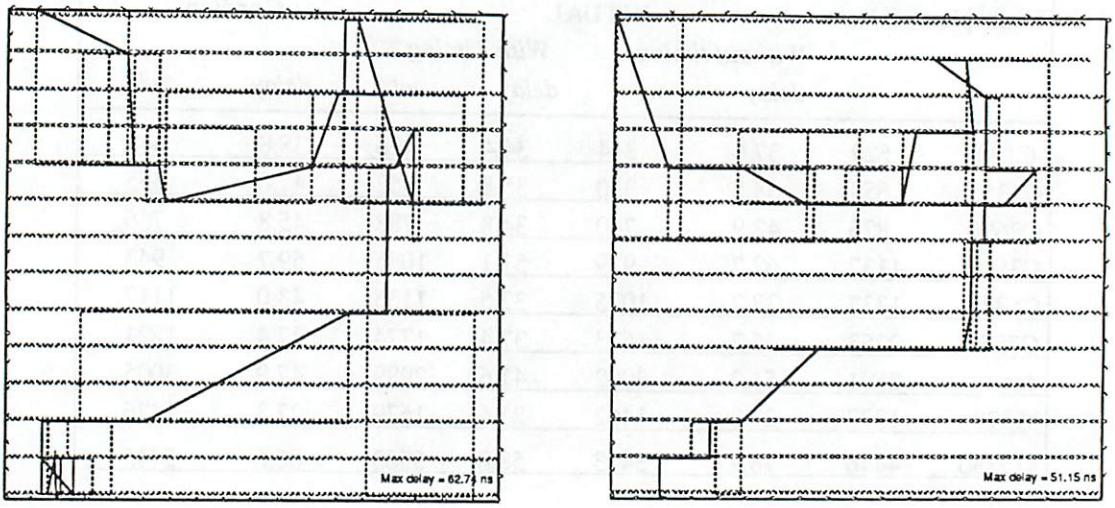


Figure 5: 1137 cell example (combinational) before and after timing optimization

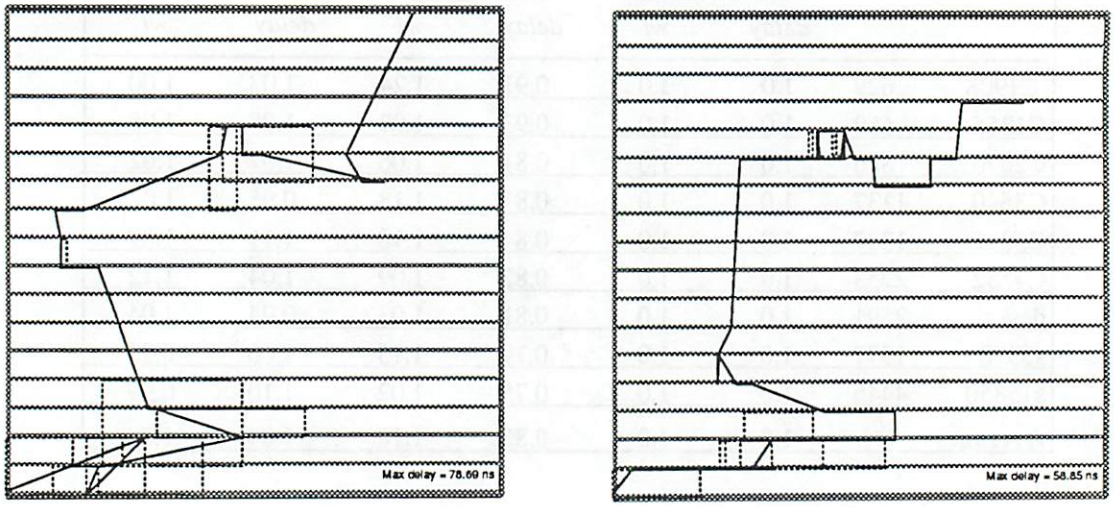


Figure 6: 4446 cell example (sequential) before and after timing optimization

tree approximation.

We observed an average of 18% improvement in the total delay over Gordian at the cost of 6% increase in the wirelength. Compared to RITUAL without timing we observed 17% improvement in the wire delay while wire length increased by 10%. RITUAL in the wirelength mode is about 4% better in the wirelength *and timing*. A further improvement of 5-10% in the wirelength, with no degradation in timing is observed when IO pad assignment is performed. Figures 5 and 6 show two large examples without and with timing optimization. In the figures, the most critical paths are plotted along with the bounding boxes for links whose bounding box is not determined by the link end points. Note that in the sequential example the critical path terminates at a latch.

12 Conclusions and Future Work

We have developed an efficient iterative technique for performance constrained placement of cell-based ICs. The technique satisfies timing constraints along with slot requirements and minimizes wirelength. It is shown to be effective on an array of standard benchmark examples. We are working on integrating the package with standard routers and results will be published in the final version of this paper.

Acknowledgements

The authors would like to acknowledge the support received for this research from SRC Grant 90-DC-008 and NSF grant MIP88-03711.

References

[1] Michael Burstein and Mary N. Youssef, Timing influenced layout design., *IEEE Proceedings of the 22nd Design Automation Conference*, pages 124--130, 1985.

[2] A. E. Dunlop, V. D. Agrawal, D. N. Deutsch, M. F. Jukl, P. Kozak, and M. Wiesel, Chip layout optimization using critical path weighting. *IEEE Proceedings of the 21st Design Automation Conference*, pages 133--136, 1984.

[3] M. Marek-Sadowska and S. P. Lin., Timing-driven placement, *IEEE International Conference on Computer-Aided Design*, ICCAD-89, pages 94--97, 1989.

[4] R. Nair, C. L. Berman, P.S. Hauge, and E. J. Yoffa., Generation of performance constraints for layout., *IEEE Trans. Computer-Aided Design*, CAD-8:860--874, August 1989.

[5] Yasushi Ogawa, Tatsuki Ishii, Yoichi Shiraishi, Hidekazu Terai, Tokinori Kozawa, Kyoji Yuyama, and Kyoji Chiba, Efficient placement algorithms optimizing delay for high-speed ecl masterslice lsi's., *IEEE Proceedings of the 23rd Design Automation Conference*, pages 404--410, 1986.

[6] S. Teig, R. L. Smith, and J. Seaton, Timing-driven layout of cell-based ic's, *VLSI System's Design*, pages 63--73, May 1986.

[7] P.K. Wolff, A. E. Ruehli, B. J. Agule, J. D. Lesser, and G. Goertzel, Power/timing: Optimization and layout techniques for lsi chips, *Journal of Design Automation and Fault-Tolerant Computing*, pages 145--164, 1978.

[8] K. C. Saraswat and F. Mohammadi, Effect of scaling of interconnections on the time delay of vlsi circuits, *IEEE Transactions on Electron Devices*, ED-29:645--650, April 1982.

[9] P. Gill, W. Murray, and M. Wright., Practical Optimization., Academic Press, New York, New York, 1989.

[10] S. Prasitjutrakul and W. J. Kubitz, Path-delay constrained floorplanning: A mathematical programming approach for initial placement, *IEEE Proceedings of the 26th Design Automation Conference*, pages 364--369, 1989.

[11] M. A. B. Jackson and E. S. Kuh., Performance-driven placement of cell-based ic's., *IEEE Proceedings of the 26th*

Design Automation Conference, pages 370--375, 1989.

[12] R. S. Tsay, E. S. Kuh, and C. P. Hsu, Proud: A sea-of-gates placement algorithm, *IEEE Design and Test of Computers*, pages 318--323, December 1988.

[13] Wilm Donath, R. J. Norman et. al , Timing driven placement using complete path delays, *IEEE Proceedings of the 27th Design Automation Conference*, pages 84-89, 1990.

[14] S. Sutanthavibul, E. Shragowitz, An Adaptive timing driven layout for high speed VLSI, *IEEE Proceedings of the 27th Design Automation Conference*, pages 90-95, 1990.

[15] M. Terai ,K. Takahashi and K. Sato, A New mincut placement for placement algorithm for timing assurance layout design, *IEEE Proceedings of the 27th Design Automation Conference*, pages 96--102 1990.

[16] I. Lin and D. Du , Performance-driven constructive placement, *IEEE Proceedings of the 27th Design Automation Conference*, pages 103-105, 1990.

[17] Shapiro, Jeremy F., Mathematical programming : structures and algorithms, New York : Wiley, c1979.

[18] Luenberger, David G., Introduction to linear and nonlinear programming, Reading, Mass., Addison-Wesley Pub. Co. [1973].

[19] J. M. Kleinhans, G. Sigl, F. M. Johannes, and K. J. Antreich, GORDIAN:VLSI placement by quadratic programming and slicing optimization, *IEEE Trans. on CAD*, Volume 10, No. 3, pages 356-365, 1991.

[20] E.L. Lawler, Combinatorial Optimization: Networks and Matroids, Holt, Rinehart and Winston, 1976.

[21] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, MIS: Multiple-level interactive logic optimization system, *IEEE Trans. on CAD*, Volume 6, No. 6, pages 1062-1081, 1987.

[22] A. Srinivasan, Algorithms for Performance-Driven Placement, UCB ERL Technical Memo M90/107, Electronics Research Laboratory, University of California, Berkeley, CA 94720.

[23] Arvind Srinivasan, Kamal Chaudhary and Ernest S. Kuh, RITUAL: An Algorithm for Performance-Driven Placement of Cell-Based ICs, Proceeding of the Third Physical Design Workshop, May 1991.