

A Negative Reinforcement Method for PGA Routing

F. D. Lewis and Chia-Chi Wang

Department of Computer Science
University of Kentucky
Lexington, Kentucky 40506

Contact: F. D. Lewis, lewis@cs.engr.uky.edu

Abstract

We present an efficient and effective method for the detailed routing of symmetrical or sea-of-gates FPGA architectures. Instead of breaking the problem into 2-terminal net collections we propose a model in which Steiner trees spanning nets of logic blocks are constructed on grids induced by the blocks. Then a new routing technique, negative reinforcement is employed to prevent nets from blocking each other. The experimental results have proven promising.

1. Introduction.

Programmable Gate Arrays (PGA's) have proven a very attractive paradigm for semi-custom VLSI design in the industrial world. Their popularity has been due primarily to the short design and fabrication times required, especially when compared to full custom design methods for VLSI circuits. Recently, the use of Field Programmable Gate Arrays (FPGA's) has emerged as a means of implementing logic circuits for Application-Specific Integrated Circuits (ASIC's) when quick turnaround, reconfigurability, and low-cost prototypes become essential .

After the specification stage, the CAD process for FPGA's proceeds through logic optimization, technology mapping, placement, and routing. A major problem in routing is keeping nets from overlapping and blocking each other. Many solutions to this problem involve specifying connections as 2-terminal nets and then employing maze-routers [Lee61] with successful use of rip-up and reroute techniques [AS89, TS88, RT86, CC88, Pal92, Nai87, SMT86]. Wire-pushing [SS87] has proven interesting as has introducing a global routing phase [AS89, TS88, OO92]. Special techniques for Symmetrical Array FPGA [BRV90, RB90, RB91] have also been examined.

Multiterminal, multinet wiring problems have been examined in general via branch and bound [YW73] and linear programming relaxation [RT91] as well as more traditional rip-up methods [LBH76] and special variations of Steiner trees [CSW89]. Even a language has been developed to describe collections of trees suitable for routing [NRT86].

In our treatment, we turn away from 2-terminal nets and propose a model for detailed routing in which Steiner spanning trees are constructed on a grid induced by the FPGA logic blocks. Then, overlaps between nets are removed in an iterative fashion with a new *negative reinforcement* technique. This provides a framework in which entire nets may be routed simultaneously as well as leading to shorter overall routing wire length.

This paper is organized as follows. Section 2 contains basic definitions of Steiner trees, grids, and tours. FPGA architectures are explored in section 3 and a routing model is developed in section 4. Properties of the routing problem and routing model are presented in section 5. Routing of single spanning trees is developed in section 6 and extended to groups of trees in section 7. Results are presented in section 8.

2. Preliminaries.

In order to develop a routing model and algorithm for detailed FPGA routing we shall need Steiner trees, weighted grids, as well as paths and tours over grids.

A *shortest rectilinear Steiner spanning tree* (or Steiner tree) over a set of points (in the plane) is a minimal length collection of vertical and horizontal lines connecting the points. According to Hanan [Han66], one of these always exists on the smallest complete rectangular grid containing the points, known as the *grid induced by the points*. Figure 1a shows an example of four points with their induced grid and figure 1c contains a shortest rectilinear Steiner spanning tree over these points.

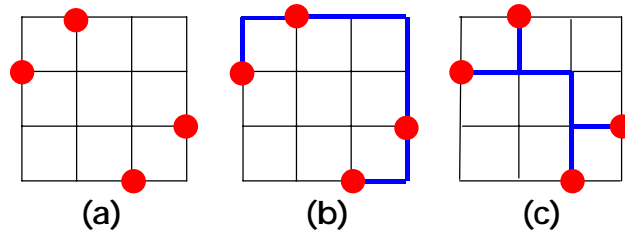


Figure 1 - Induced Grid and Spanning Trees

Rectilinear minimum spanning trees (MST's) also exist on the induced grid. One is depicted in figure 1b. A result due to Hwang [Hwa76] assures us that they are no larger than one and a half times the size of the shortest rectilinear Steiner tree spanning the points. And, of course, no spanning tree can be smaller than one half the perimeter of the induced grid.

Heuristic algorithms for constructing Steiner trees are often necessary since the problem is NP-complete [GJ77] for more than three points. A survey of these approximation methods appears as part of [HRW92].

A *grid* is a rectilinear collection of edges and an *edge* runs between *grid intersections*. Note that the induced grids mentioned above are *complete* in that inside the perimeter all edges are included. Not all of the grids we shall use below are complete.

A *tour* of a set of points is a connected sequence of edges on the induced grid, which, when traversed, takes one through all of the points. It is a special type of *path* which is merely a connected sequence of edges.

3. FPGA Architectures.

Several companies have provided various architecture types for FPGA's in recent years [BFR92]. Two commercially available FPGA families of interest here are the Symmetrical Array or Island-Type (Xilinx and QuickLogic) and the Sea-of-Gates (Plessey, Algotronix, and Concurrent). Illustrations of both of these appear in Figure 2.

Each is an array of Configurable Logic Blocks (CLB's). In the Symmetrical Array architecture the red blocks are CLB's and connect (through the circles) to a grid of interconnect which contains switchboxes (the blue boxes) at intersections. The boxes in the Sea-of-Gates FPGA also form an array of CLB's. Each is connected locally to its neighbors.

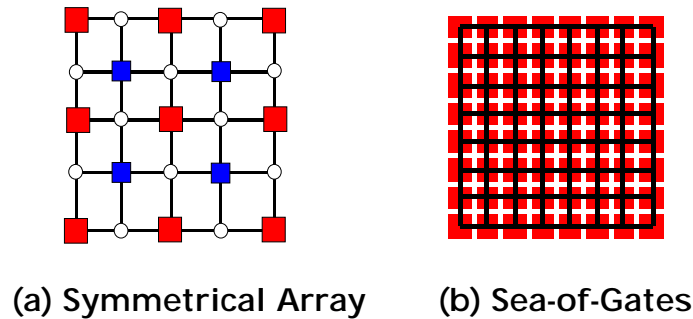


Figure 2 - Two FPGA Architectures

Connections through CLB's (usually through multiplexers), and over the top of the array (with a dense interconnect resource) may also be made.

4. The Routing Model.

In order to formulate a model which applies to both of the architectures featured in the last section, we must examine the general routing problem. With this in mind we depict our routing area as a grid and place the groups of CLB's to be connected as points at intersections on the grid. A group of CLB's which are to be connected is called a *net*. An example featuring two three point nets and their induced grid is displayed in figure 3a.

Steiner trees (one connecting each set of points or net) on the grid induced by the union of the points in the nets shall provide our routing. Because of physical wiring constraints, we need some rules which mandate exactly which edges of the grid may be used to build these Steiner trees. We shall not allow parts of a tree spanning a net to cross any of the points (CLB's) from other nets. Thus each tree is *restricted* to a portion of the induced grid. Figure 3b shows the part of the grid that can be used to build the tree connecting the red points. We call this the *grid restricted to the red net*. Figure 3c displays the grid restricted to the blue net.

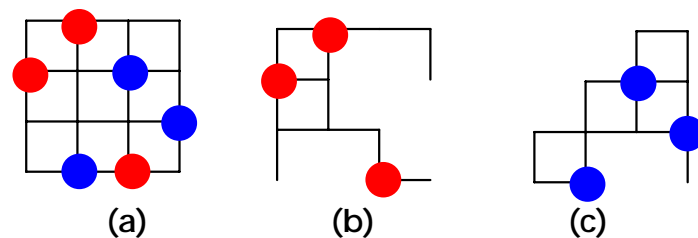


Figure 3 - Nets and Grids Restricted to Nets

Given a grid induced by several nets, we may now define these restricted grids formally.

Definition. The *grid restricted to a net* contains all of the grid edges which are not connected to any other nets.

For pairs of nets the notation we shall use is defined as follows. If G is the rectangular grid induced by the nets containing the sets of points P_a and P_b , then the two restricted grids are:

$$G_a = G - \{\text{edges connected to } p_i \in P_b\}$$

$$G_b = G - \{\text{edges connected to } p_j \in P_a\}$$

Other conventions and constraints apply to our routing model. We shall allow lines in two trees to *cross* at a grid intersection (since we have at least 2-layer routing) or to *turn* at an intersection (this is knock-knee routing) as pictured in (b) and (c) of figure 4. We do not allow *overlaps* as shown in 4a, though this constraint may be relaxed if edges on the routing grid are allowed to have *capacities* greater than one.

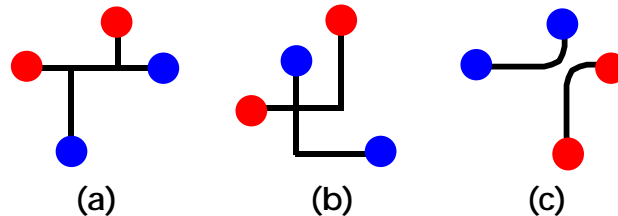


Figure 4 - Routing Conventions

Below we state this precisely and simply for the case involving two nets containing the sets of points P_a and P_b . Our routing problem is now just a multiple Steiner spanning problem on a grid.

Definition. The *nonoverlapping multiple rectilinear Steiner spanning problem* for two nets containing the sets of points P_a and P_b is the construction of the smallest pair of rectilinear Steiner spanning trees, $T_a \in G_a$ which spans P_a , and $T_b \in G_b$ which spans P_b such that $T_a \cap T_b = \emptyset$.

Note that this is the *optimum solution*, that is, the *smallest* Steiner forest (one tree per net) which spans the points in the nets and contains no overlaps. Since there may be no nonoverlapping solution in many cases, we shall also be interested in the problem where overlaps are minimized.

5. Properties of the Problem and Model.

Before attempting to solve this multinet routing problem we examine our routing model. As with the single net case, we shall in most cases be restricted

to heuristic algorithms because of the following result which also holds for the minimum overlap problem.

Theorem 1. *The nonoverlapping multiple rectilinear Steiner spanning problem is NP-complete.*

Proof. If none of the points in the nets share coordinates, then this is merely the rectilinear Steiner spanning problem. Since that is NP-complete [GJ77], so is the nonoverlapping version.

A question of primary importance is whether there is any solution to a particular problem. There are cases for which there is no possible way to route the nets on their restricted grids without overlaps. Consider the two nets in figure 5a which have the restricted grids shown in figures 5b and 5c. It is obvious that there is no way to route both at the same time on these grids with no overlaps because the routing for each is required to traverse the central square of their restricted grids.

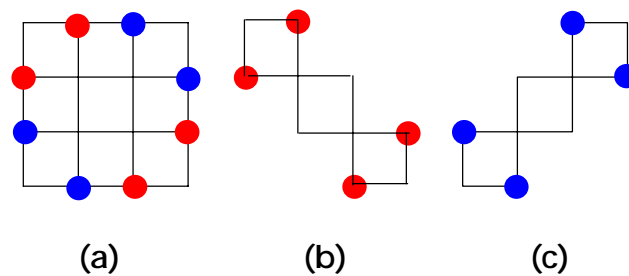


Figure 5 - Nets not Routeable Simultaneously

Thus, in several simple cases it is obvious that there is no solution to the multinet rectilinear Steiner spanning problem. Two are:

- One single connected component of the restricted grid does not contain all of the points in a set.
- Two restricted grids share an *articulation area* (such as the center square in both restricted grids of figure 5) which must be traversed by each and will always result in an overlap.

The closest we come to a general symptom for no solution is the following theorem.

Theorem 2. *There is no solution for the nonoverlapping multiple rectilinear Steiner spanning problem (over two nets) if and only if every pair of net-tours (one on each set of points) on the restricted grids contains an overlap.*

Proof. If there is a solution to the problem then there is a disjoint pair of spanning trees for the sets of points. A depth-first traversal of the tree edges produces a tour of the points.

If there are two tours (one on each set of points) which do not overlap, these form the spanning trees needed for a solution to the problem when all cycles are removed.

This result provides intuition that determining if there is a solution to the nonoverlapping Steiner problem is no easier than many other path problems and thus is NP-complete as well. For this reason, we develop algorithms to solve the *minimum overlap* problem. For this problem we easily know when there is a solution.

Theorem 3. *There is a solution for the multiple rectilinear Steiner spanning problem if and only if each set of points is in one connected component of its own restricted grid.*

Proof. If there is a solution it consists of two spanning trees. These trees exist on connected portions of the grid which span the points. And, if the nets are on connected components of the grid, then there are trees which span them.

Our last query concerns the optimality of a solution. In the single tree case we were able to relate solutions to the size of minimum spanning trees using Hwang's theorem [Hwa76] and guarantee an size range, namely somewhere between the minimal spanning tree size and two-thirds of it. Here we are not so fortunate. Consider the problem depicted in figure 6. The minimum length Steiner spanning forest for the nets is shown in figure 6a, while figures 6b and 6c provide the two minimum spanning trees over the nets on their restricted grids. This confirms that the sum of the sizes of the minimal spanning trees *does not* form an upper bound on the length of the Steiner spanning forest for the nonoverlapping Steiner spanning problem.

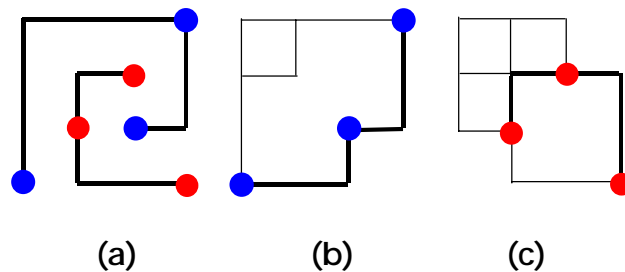


Figure 6 - Steiner Forest and MST's

We do have two lower bounds. One is merely half the sum of the perimeters of the smallest rectangles enclosing the nets and the other is the sum of the smallest Steiner trees built without regard to overlaps. This means that we still have Hwang's lower bound of two-thirds the minimal spanning tree size if these spanning trees are constructed on the induced rather than the restricted grids.

6. Single Steiner Trees.

Before we are able to route several nets at once, we must develop a method for routing one net. This means generating Steiner trees on the restricted grids for nets. We elect to use an algorithm based upon Kruskal's greedy algorithm for minimal spanning trees [Kru56]. Thus our algorithm is similar in spirit to other Kruskal-based procedures [BC85, LV91] which have been formulated for use on complete induced grids rather than the restricted grids which we must use.

Using the restricted grid means that we must route in the *presence of obstacles* [RLW89, WWS87]. This makes distance computation more difficult since paths between points must be discovered rather than just computing distances from coordinates. We also note that there is a slight increase in the time complexity over some of the very efficient methods and find that there are less dramatic differences in size between minimum spanning trees and Steiner trees than in the case without obstacles.

Following Kruskal, we combine Steiner trees for subnets a greedy manner. We begin with each point as a tree and then connect the pair of closest trees to form a new tree. Then we determine the distance from the edges of this new tree to the remaining trees. This continues until only one tree remains. This is illustrated in figure 7 where first the upper left corner points are connected, then the bottom right ones, and at last, both trees are joined.

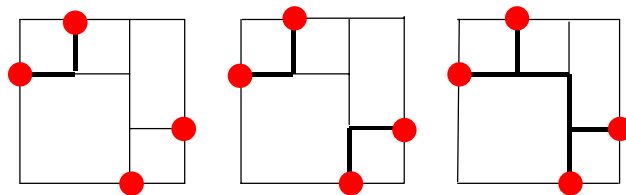


Figure 7 - Building a Steiner Tree

The algorithm appears below in figure 8.

BuildTree(P, G, T)
PRE: $P = (p_1, \dots, p_n)$ = set of points = net
 G = grid containing all points in P
POST: T = Steiner tree spanning P on grid G

trees = {1, ..., n};
 $T_1 = \emptyset; \dots; T_n = \emptyset$
 $S_1 = \{p_1\}; \dots; S_n = \{p_n\}$
 $D[i, k]$ = shortest distance on G between S_i and S_k
 repeat n-1 times
 Select $i, k \in$ trees with minimum $D[i, k]$
 Path = shortest path on G between S_i and S_k
 $S_i = S_i \cup S_k \cup \{\text{grid intersections on Path}\}$
 $T_i = T_i \cup T_k \cup \text{Path}$
 trees = trees - {k}
 Update $D[i, m]$ for all $m \in$ trees
 $T =$ the only remaining T_i

Figure 8 - Single Steiner Tree Algorithm

Since we are building a tree on a grid which is not complete there can be no relation between the best and worst-case sizes of the tree. We can however bound the size and prove the algorithm's correctness.

Theorem 4. *The BuildTree algorithm constructs a Steiner spanning tree over the set of points in the net which is no larger than the minimal spanning tree over the points.*

A note on the complexity is in order. Since we are not routing on a complete grid, we must detect paths rather than calculate distances. Using wavefront propagation techniques in the spirit of the maze crawling algorithm of Lee [Lee61] we might search the entire grid. This means, in the worst case, $O(n^2)$ time for updating the distance matrix $D[i,j]$ and recording the paths between trees. Thus our BuildTree algorithm has worst case time complexity $O(n^3)$.

Complete details for all aspects of this algorithm are found in [Pon93].

7. Simultaneous Tree Generation.

We begin by developing an algorithm to route two nets of points at the same time and then extend it to arbitrarily large numbers of nets. Figure 9 contains the main algorithm where we check to see if there is any possible routing and if so, find one. Most of the variables should be familiar (trees, grids, and nets or

sets of points) and *maxtries* is a parameter set by the user to bound the number of times we attempt to remove overlaps from the routing.

```

input( $P_a, P_b, \text{maxtries}$ )
Build induced grids  $G_a$  and  $G_b$  restricted to  $P_a$  and  $P_b$ 
if ( $P_a$  are connected on  $G_a$ ) and ( $P_b$  are connected on  $G_b$ )
  then Route( $P_a, P_b, G_a, G_b, \text{maxtries}, T_a, T_b$ )
  else report routing failure

```

Figure 9 - Main Routing Algorithm

Our strategy for routing the two-net case is to first construct Steiner spanning trees on the restricted grids for each net. Next, we check to see if they overlap. If so, we penalize the use of the overlapped edge by pretending that its length is doubled and reconstructing one of the trees. This continues until there are no overlaps, or, until we reach a limit (named *maxtries*) placed on the number of attempts to correct overlaps. If this limit is reached, the pair of trees previously constructed with the minimum amount of overlap is selected.

An example appearing in figure 10 demonstrates the algorithm. In 10a both trees overlap on the two grid edges in the yellow area. The grid edges comprising the overlap are assigned larger weights (their length is doubled) and the tree spanning the blue points is redrawn in 10b.

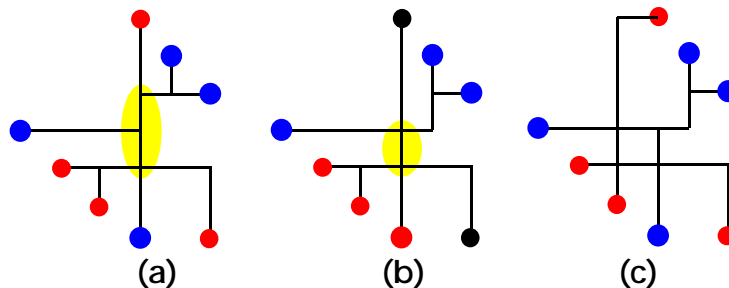


Figure 10 - Routing Two 4-Point Nets

Since there is still an overlap, the weight of the edge in the yellow area is increased and the tree spanning the red points is redrawn in 10c. At last there is no overlap.

Trees are of course constructed using the algorithm of figure 8 and checking overlaps between trees is accomplished by sorting the grid edges which make up the trees and comparing them. Since a bucket sort may be used, the complexity is the same as the number of edges in the trees. (And even this can be reduced if we omit the edges connected to points since they're not in the other restricted grid.) Thus the complexity of algorithm is dominated by the time used to build Steiner spanning trees.

Our *negative reinforcement* routing algorithm for pairs of nets is presented in figure 11.

```

Route( $P_a, P_b, G_a, G_b, \text{maxtries}, T_a, T_b$ )
  PRE :  $G_a$  and  $G_b =$  induced grids restricted to  $P_a$  and  $P_b$ 
  POST:  $T_a$  and  $T_b =$  Steiner Trees spanning  $P_a$  and  $P_b$  on  $G_a$  and  $G_b$ 

  tries = 0
  Construct Steiner tree  $T_a$  spanning  $P_a$  on  $G_a$ 
  repeat
    Construct Steiner tree  $T_b$  spanning  $P_b$  on  $G_b$ 
    CheckOverlaps( $T_a, T_b$ )
    if (overlaps exist) then
      Adjust length of overlapped segments in  $G_a$ 
      Construct Steiner tree  $T_a$  spanning  $P_a$  on  $G_a$ 
      CheckOverlaps( $T_a, T_b$ )
      if (overlapped segments exist) then
        Adjust length of overlapped segments in  $G_b$ 
      tries ++
  until (no overlaps remain) or (tries > maxtries);

```

Figure 11 - Simultaneous Routing Procedure

Extending the algorithm to the case involving k nets involves iteration over the sets. Our technique is to check for overlaps involving all of the trees *except* that about to be constructed. Consider the code fragment in figure 12.

```

for  $i = 1$  to  $k$  do
  CheckOverlaps( $T_1, \dots, T_{i-1}$ )
  tries = 0
  while (overlaps exist) and (tries  $\leq$  maxtries) do
    Adjust length of overlapped segments in  $G_{i-1}$ 
    Construct Steiner tree  $T_{i-1}$  spanning  $P_{i-1}$  on  $G_{i-1}$ 
    CheckOverlaps( $T_1, \dots, T_{i-1}$ )
    if (overlapped segments exist) then
      for every  $T_j$  ( $1 \leq j < i-1$ ) with overlapped edges do
        Adjust length of overlapped segments in  $G_j$ 
        Construct Steiner tree  $T_j$  spanning  $P_j$  on  $G_j$ 
      CheckOverlaps( $T_1, \dots, T_{i-1}$ )
    tries++
    Adjust length of overlapped segments in  $G_i$ 
  Construct Steiner tree  $T_i$  spanning  $P_i$  on  $G_i$ 

```

Figure 12 -Multinet Routing Algorithm

In this algorithm note that the last tree constructed is rebuilt if overlaps still exist and that if the limit on iterations has been reached a tree is still constructed.

As with the two net case, constructing Steiner trees dominates the complexity of the algorithm. Since up to k trees can be built each time we repeat the inner loop, our complexity is $O(k^2n^3)$.

Another method is to construct the tree at the beginning of the loop, check for overlaps involving *all* trees, and continue on to the next tree. This variation was implemented and was not found to be significantly different from the algorithm in figure 12.

8. Empirical Results.

The algorithm was implemented in C and run on a Sequent Symmetry S81 Dynix 3.0 machine with randomly generated data sets. Three parameters were varied:

- the number of nets in each data set,
- the number of points (n) in each net, and
- the size of the grid induced by points in the nets.

The latter is necessary since we wish points in different nets to share coordinates. If they do not, then there are shortest Steiner trees spanning each net over the grids *they themselves* induce due to a result of Hanan's [Han66]. Thus grids smaller than $2n$ by $2n$ are needed.

Three induced grid sizes ($1.75n$, $1.5n$, and $1.25n$) were utilized for data sets involving groups of n point nets. Note that using a $1.5n$ by $1.5n$ induced grid size for a pair of nets forces half of the points from each net to share x -coordinates with points from the other net and half from each to share y -coordinates. Thus our three grid sizes provide 25%, 50%, and 75% coordinate sharing between nets.

The negative reinforcement algorithm of figure 11 was run on data sets composed of pairs of 5 to 50 point nets. Each data set contained 10 pairs of nets. Sizes of the minimal spanning trees, the initial Steiner spanning trees, and the final Steiner trees over each net were recorded and compared.

For each net, the sum of the sizes of the Steiner spanning trees was compared to the combined size of minimal spanning trees over the nets. Table 1 shows the amount of improvement over minimal spanning trees achieved by the negative reinforcement algorithm on pairs of nets for n point data sets using the three induced grid sizes.

| n | 1.75n Grid | | 1.5n Grid | | 1.25n Grid | |
|-----|------------|------|-----------|------|------------|------|
| | First | Last | First | Last | First | Last |
| 5 | 7.05 | 6.95 | 6.55 | 3.23 | 4.86 | 1.17 |
| 10 | 6.99 | 6.86 | 6.65 | 6.07 | 6.82 | 5.15 |
| 20 | 5.59 | 5.59 | 5.61 | 5.33 | 5.62 | 4.79 |
| 30 | 6.06 | 5.89 | 6.58 | 6.24 | 5.10 | 5.00 |
| 50 | 5.57 | 5.54 | 5.46 | 5.37 | 5.33 | 5.06 |
| Ave | 6.25 | 6.17 | 6.17 | 5.25 | 5.55 | 4.23 |

Table 1 - Percent Improvement for 2-Net Data Sets

In the columns labeled 'First' the average percent improvement of the initial Steiner trees over the minimal spanning trees on the restricted grids is indicated, while the 'Last' columns show the average percent improvement after iterating until there were no overlaps between the pairs of nets.

As expected, denser grids provided cases for which less improvement was possible. But, non overlapping Steiner spanning trees were found for every data set. Most data sets required no more than three iterations to remove overlaps, and the worst data set (1.25n grid with 5 points) needed six iterations.

Table 2 and table 3 provide the average improvement for three and five net data sets on the same size induced grids.

| n | 1.75n Grid | | 1.50n Grid | | 1.25n Grid | |
|-----|------------|------|------------|------|------------|------|
| | First | Last | First | Last | First | Last |
| 5 | 6.60 | 6.07 | 4.45 | 3.37 | 6.10 | 0.61 |
| 10 | 6.63 | 6.51 | 5.99 | 5.07 | 5.16 | 3.78 |
| 15 | 5.23 | 4.88 | 5.66 | 4.94 | 5.45 | 3.94 |
| 20 | 5.61 | 5.42 | 5.14 | 4.65 | 6.05 | 4.65 |
| Ave | 6.02 | 5.72 | 5.31 | 4.51 | 5.69 | 3.24 |

Table 2 - Percent Improvement for 3-Net Data Sets

| n | 1.75n Grid | | 1.50n Grid | | 1.25n Grid | |
|-----|------------|------|------------|-------|------------|------|
| | First | Last | First | Last | First | Last |
| 5 | 6.48 | 4.33 | 4.25 | -1.77 | 2.18 | |
| 10 | 6.17 | 5.89 | 5.78 | 4.82 | 6.02 | 2.41 |
| 15 | 5.56 | 5.32 | 4.71 | 3.33 | 5.70 | 3.01 |
| 20 | 4.84 | 4.70 | 5.61 | 5.06 | 5.53 | 1.14 |
| Ave | 5.76 | 5.06 | 5.09 | 2.86 | 4.86 | 2.19 |

Table 3 - Percent Improvement for 5-Net Data Sets

These data sets were packed even more densely than the pairs of nets, and so provided even less pleasing results. In fact, we could generate no data sets on the $1.25n$ grid for the five-net case because articulation areas always appeared.

Reduction of overlaps was very successful. In all of the test cases, *only eight resulted in spanning tree forests which contained overlaps!* The number of iterations in the three-net case was less than usually less than five with one at seven, and the five-net data sets normally required no more than 10 iterations with a maximum of 18 iterations.

9. Conclusion.

We have presented an efficient and effective method for the detailed routing of symmetrical or sea-of-gates FPGA architectures. Part of this is due to the fact that it is based upon simultaneous Steiner spanning trees rather than sequential routing of 2-terminal net collections. In the proposed model, Steiner trees spanning nets of logic blocks are constructed on grids induced by these blocks by an algorithm guaranteed to be at least as good as an MST algorithm applied to the modified grids.

The new routing technique, negative reinforcement, is employed to prevent nets from blocking each other. Experiment have been successful in providing nonoverlapping routings in 97% of the test cases.

References

- AS89** Adams, G. D., and Séquin, C. H. "Template Style Considerations for Sea-of-Gates Layout Generation." *Proc. of 26th ACM/IEEE DAC*, 1989, 31-36.
- BC85** Bern, M. W. and deCarvalho, V. "A Greedy Heuristic for the Rectilinear Steiner Tree Problem." *Tech. Rpt. Comp. Sci. Div.*, UC Berkeley, 1985.
- BFR92** Brown, S. D., Francis, R. J., Rose, J. and Vranesic, Z. G. *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, Boston, 1992.
- BRV90** Brown, S. D., Rose, J., and Vranesic, Z. G. "A Detailed Router for Field-Programmable Gate Arrays." *Proc. IEEE ICCAD*, 1990, 382-385.
- CC88** Chakraverti, A. and Chung, M. J. "Routing Algorithm for Gate Array Macro Cells." *Proc. 25th ACM/IEEE DAC*, 1988, 658-662.
- CSW89** Chiang, C., Sarrafzadeh, M., and Wong, C. K. "A Powerful Global Router Based on Steiner Min-Max Trees." *Proc. IEEE ICCAD, 1989, 2-5*.
- GJ77** Garey, M. R. and Johnson, D. S. "The Rectilinear Steiner Problem is NP-Complete." *SIAM J. Appl. Math.*, vol. 32, 1977, 826-834.
- Han66** Hanan, M. "On Steiner's Problem with Rectilinear Distance." *SIAM J. Appl. Math.*, vol. 14, no. 2, 1966, 255-265.
- Hwa76** Hwang, F. K. "On Steiner Minimal Trees with Rectilinear Distance." *SIAM J. Appl. Math.*, vol. 30, 1976, 104-114.
- HRW92** Hwang, F. K., Richards, D. S., and Winter, P. *The Steiner Tree Problem. Annals of Discrete Mathematics*, no. 53, North Holland, 1992.
- Kru56** Kruskal, J. B. "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem." *Proc. of the AMS*, vol. 7, 1956, 48-50.
- Lee61** Lee, C. Y. "An Algorithm for Path Connections and Its Applications." *IRE Trans. on Electronic Computers*, vol. EC-10, Sept 1961, 346-365.
- LBH76** Lee, J. H., Bose, N. K., and Hwang, F. K. "Use of Steiner's Problem in Suboptimal Routing in Rectilinear Metric." *IEEE Trans. on Circuits Syst.*, vol. CAS-23, 1976, 470-476.
- LV91** Lewis, F. D. and Van Cleave, N. "Correct and Provably Efficient Methods for Rectilinear Steiner Spanning Tree Generation," *Proceedings of the First Great Lakes Computer Science Conference*, and *Springer-Verlag Lecture Notes*, vol. 507, 1991.
- Nai87** Nair, R. "A Simple Yet Effective Technique for Global Wiring." *IEEE Trans. on CAD*, vol. CAD-6, no. 2, March 1987, 165-172.
- NRT86** Ng, A. P-C., Raghavan, P., and Thompson, C. D. "A Language for Describing Rectilinear Steiner Tree Configurations." *Proc. 23rd ACM/IEEE DAC*, 1986, 659-662.
- OO92** Okuda, R. and Oguri, S. "An Efficient Routing Algorithm for SOG Cell Generation on a Dense Gate-Isolated Layout Style." *Proc. 29th ACM/IEEE DAC*, 1992, 676-681.

- Pal92** Palczewski, M. "Plane Parallel A* Maze Router and its Application to FPGA's." *Proc. 29th ACM/IEEE DAC*, 1992, 691-697.
- Pon93** Pong, W. C.-C. "Steiner Trees and CAD Algorithms for VLSI Systems." *PhD Dissertation*, University of Kentucky, 1993.
- RT91** Raghavan, P., and Thompson, C. D. "Multiterminal Global Routing: A Deterministic Approximation Scheme." *Algorithmica*, vol. 6, 11991, 73-82.
- RLW89** Rezende, P. J., Lee, D. T., and Wu, Y. F. "Rectilinear Shortest Paths in the Presence of Rectangular Barriers." *Discrete & Comput. Geom.*, vol. 4, 1989, 41-53.
- RB90** Rose, J. and Brown, S. "The Effect of Switch Box Flexibility on Routability of Field Programmable Gate Arrays." *Proc. 1990 CICC*, 1990, 27.5.1-27.5.4.
- RB91** Rose, J. and Brown, S. "Flexibility of Interconnection Structures in Field-Programmable Gate Arrays." *IEEE J. of Solid State Circ.*, vol. 26, no. 3, March 1991, 277-282.
- RT86** Rowson, J. and Trimberger, S. "Gate Array Macro Layout Automation." *Proc. 1986 IEEE ICCAD*, 1986, 448-451.
- SS87** Shin, H. and Sangiovanni-Vincentelli, A. "A Detailed Router Based on Incremental Routing Modifications: Mighty." *IEEE Trans. on CAD*, vol. CAD-6, no. 6, November 1987, 942-955.
- SMT86** Suzuki, K., Matsunaga, Y., Tachibana, M. and Ohtsuki, T. "A Hardware Maze Router with Application to Interactive Rip-Up and Reroute." *IEEE Trans. on CAD*, vol. CAD-5, no. 4, October 1986, 466-476.
- TS88** Tzeng, P. and Séquin, C. H. "Codar: A Congestion-Directed General Area Router." *Proc. of 1988 IEEE ICCAD*, 1988, 30-33.
- YW73** Yang, Y. Y. and Wing, O. "On a Multinet Wiring Problem." *IEEE Trans. Circuit Theory*, vol. CT-20, 1973, 250-252.
- WWS87** Wu, Y. F., Widmayer, P., Schlag, M. D. F., and Wong, C. K. "Rectilinear Shortest Paths and Minimum Spanning Trees in the Presence of Rectilinear Obstacles." *IEEE Trans. on Comput.*, vol. C-36, 1987, 321-331.