PROUD: A FAST SEA-OF-GATES PLACEMENT ALGORITHM

by

Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu

Memorandum No. UCB/ERL M87/79

20 November 1987

PROUD: A FAST SEA-OF-GATES PLACEMENT ALGORITHM

by

Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu

Memorandum No. UCB/ERL M87/79

20 November 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

TITLE PAGE

PROUD: A FAST SEA-OF-GATES PLACEMENT ALGORITHM

by

Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu

Memorandum No. UCB/ERL M87/79

20 November 1987

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

PROUD: A Fast Sea-Of-Gates Placement Algorithm

*Ren-Song Tsay, Ernest S. Kuh, and Chi-Ping Hsu**

Department of EECS and the Electronics Research Laboratory
University of California
Berkeley, CA 94720
(415) 642-4325

Abstract

We present a fast and effective placement algorithm which takes advantage of inherent sparsity in the connectivity specification. It solves repeatedly sparse linear equations by the SOR method in a top-down hierarchy. The algorithm has been implemented; for a triple-metal-layer 100K sea-of-gates design with 26,000 instances, it takes 50 minutes on a VAX 8650 and yields excellent results.

* Chi-Ping Hsu is with Hughes Aircraft Company, 500 Superior Avenue, Newport Beach, Ca., 92663.

1. Introduction

As Application Specific Integrated Circuits (ASIC) become more popular and the gate count on a chip gets increasingly larger, the quality and speed of automatic layout algorithms need to be readdressed. It is well-known that the problem of cell placement is especially crucial to the final outcome of the design [1]. For a sea-of-gates chip with 100K or more gates, conventional deterministic methods are no longer suitable because sparsity of the connectivity specification ought to be considered. Random algorithms such as simulated annealing could take a forbidden amount of time in reaching an acceptable solution.

In this paper we introduce a hierarchical method of placement which takes full advantage of the sparsity inherent in the placement specification. The method uses the quadratic placement formulation first proposed by Hall [2], however we do not find the eigenvalues and eigenvectors of a large matrix. It depends on Cheng's and Kuh's concept of resistive network optimization in [3], but it bypasses the slot constraints and employs a simpler partitioning scheme. Our method takes the I/O pad specification as input and solves successively *linear* sparse equations. From the result of several real chips, the quality of the placement is excellent. It is superior to that obtained by TimberWolf 3.2 [4] and is comparable with that of TWOLF 405 [5]. The run time complexity of our method is $O(n \log^2 n)$ and the memory space complexity is linear. For a 100K sea-of-gates chip with 26K modules (cells), the run time is about 50 minutes on a VAX 8650 (6 MIPs machine) and the memory requirement is about 11 megabytes.

In Section 2, we give the formulation of the global placement problem which is used for initial placement and review pertinent concepts from earlier work [2, 3]. Section 3 deals with the next step, that is, two-way partitioning and iterations. Together they constitute one cycle of the top-down hierarchy. The pseudo code of our method is given in Section 4. Section 5 gives the complexity of the algorithm. Section 6 describes some experimental results. We conclude the paper in Section 7 and indicate some future directions.

2. The Global Placement

The global placement is solved based on the two usual assumptions that all modules are *point* modules and all nets are two-pin nets. Thus multi-pin nets must be preprocessed and replaced with two-pin nets. Let c_{ij} be the connectivity between module i and module j, e.g., the number of nets connecting the two modules. Thus a symmetric connectivity matrix $C = \left[c_{ij}\right]$ is introduced with $c_{ii} = 0$. As in Refs. [2, 3] we use the sum of the squared wire lengths as the objective function. Let l_{ij} be the distance between module i and module j, then with n modules, the objective function is

$$L = \frac{1}{2} \sum_{i,j=1}^{n} c_{ij}\, l_{ij}^2 \qquad (1)$$

$$= \frac{1}{2} \sum_{i,j=1}^{n} \left[c_{ij}\,(x_i - x_j)^2 + c_{ij}\,(y_i - y_j)^2\right]$$

where (x_i, y_i) represents the coordinate of module i.

Next, a modified connectivity matrix **B** is defined:

$$\mathbf{B} = \mathbf{D} - \mathbf{C} \tag{2a}$$

where **D** is a diagonal matrix with

$$d_{ii} = \sum_{j=1}^{n} c_{ij} \tag{2b}$$

It is easily shown that Eq. (1) can be rewritten as:

$$L(\mathbf{x}, \mathbf{y}) = \mathbf{x}^T \mathbf{B}\,\mathbf{x} + \mathbf{y}^T \mathbf{B}\,\mathbf{y} \tag{3}$$

where **x** and **y** are n-vectors which specify the coordinates of n modules. The usual placement problem is then to minimize L subject to the "slot" constraints, that is, all point modules are placed on an evenly spaced, two-dimensional grid. The following discussion summarizes the background material as well as our method for global placement.

1. In [2], the linear placement problem is treated first, i.e., only $\mathbf{x}^T \mathbf{B}\,\mathbf{x}$ in the objective function L is considered. A first order "slot" constraint, $\mathbf{x}^T \mathbf{x} = 1$, is used. This leads to the Lagrangian multiplier formulation with

$$L' = \mathbf{x}^T \mathbf{B}\,\mathbf{x} - \lambda \mathbf{x}^T \mathbf{x}$$

and the solution amounts to finding the least nonzero eigenvalue and its associate eigenvector of the matrix **B**. Since the method only takes into account the first order constraint, the solution does not satisfy the "slot" constraint, and it can only be served as a first order approximation to the final placement.

If we use the two least nonzero eigenvalues, we can find an approximation to the two-dimensional placement problem with the y-dimension coordinate obtained from the eigenvector associated with the second smallest nonzero eigenvalue. One main difficulty with this approach is that the calculation of the eigenvalues and eigenvectors of a large matrix, even for a sparse matrix, is not a simple task. Furthermore, the solution forms only a rough initial placement; to be useful, other methods must be devised as a followup.

2. In [3], higher order constraints are derived to guarantee that modules be forced onto slots. However, the nonlinear programming problem so formulated is difficult, if not impossible, to solve. By using only a first-order linear constraint, Cheng and Kuh introduced a resistive network model to obtain the initial solution explicitly from the well-known Kuhn-Tucker conditions [6]. It is then followed by relaxation and successive partitioning to obtain the final solution. Our present method takes advantage of the resistive network concept, but departs from [3] in that we ignore all constraints. Instead of solving the optimization problem directly, we solve a linear set of equations by the SOR method, a generalized Gauss-Seidel method [7].

3. Consider the electric network analogy of the placement problem. The one-dimensional objective function $\mathbf{x}^T \mathbf{B}\,\mathbf{x}$ can be interpreted as the power dissipation of an n node linear resistive network with **x** corresponding to an n-vector whose components represent the voltage at the nodes. In the remainder of this section we use **x** to designate either the coordinate or the voltage vector. The modified connectivity matrix **B** is exactly the indefinite

admittance matrix of the n-node resistive network with $-b_{ij}$ equal to the conductance between node i and node j. The placement problem is then equivalent to that of choosing the node voltages of the n-node network for which the power dissipation is a minimum. In electric network theory it is well-known that minimum power dissipation is implied by the two Kirchhoff Laws. Thus we can reformulate our linear placement problem in terms of a linear resistive network problem provided that we include the I/O pad specifications. This allows us to model the I/O pads as fixed voltage sources applied to the network, and the coordinates of the movable modules are then the node voltages to be determined.

4. In Fig. 1 we show a network where nodes 1 to m represent movable modules and nodes $m+1$ to n represent fixed modules with voltages specified by the I/O pads. Then the n-port network equation can be written as follows:

$$\mathbf{B}_{11}\,\mathbf{x}_1 + \mathbf{B}_{12}\mathbf{x}_2 = 0 \tag{4a}$$

$$\mathbf{B}_{21}\,\mathbf{x}_1 + \mathbf{B}_{22}\mathbf{x}_2 = \mathbf{i}_2 \tag{4b}$$

where $\mathbf{B}_{11}$, $\mathbf{B}^T{}_{12}=\mathbf{B}_{21}$, and $\mathbf{B}_{22}$ are the familiar submatrices of the node admittance matrix of the network. $\mathbf{x}_1$ is the voltage vector of dimension m to be determined, where m is the number of movable modules. $\mathbf{x}_2$ is the voltage source vector, and $\mathbf{i}_2$ represents the current vector flowing into the $n-m$ terminals from the sources shown in Fig. 1. Eq. (4a) is the key equation which we depend on, and it is rewritten as:

$$\mathbf{A}\,\mathbf{x}_1 = \mathbf{b} \tag{5}$$

where

$$\mathbf{A} \triangleq \mathbf{B}_{11} \text{ and } \mathbf{b} = -\mathbf{B}_{12}\mathbf{x}_2 \tag{6}$$

are given. Therefore we have converted the optimization problem into a linear algebraic problem. Since $\mathbf{A}$ is sparse, usually 0.1 percent to 1 percent, familiar sparse matrix techniques can be used to solve for $\mathbf{x}_1$.

5. We have chosen the Successive-Over-Relaxation (SOR) method to solve the sparse linear equations. The method is summarized below:
Let

$$\mathbf{A} = \mathbf{\Lambda}\,(\mathbf{L}+\mathbf{1}+\mathbf{U})$$

where $\mathbf{\Lambda}$ is a diagonal, positive definite matrix, $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix. The vector $\mathbf{x}_1$ is solved iteratively by the following recursive formula:

$$\mathbf{x}_1\,(k+1) = \mathbf{M}\,\mathbf{x}_1\,(k) + \mathbf{a}$$

where

$$\mathbf{M} = (\mathbf{1}+w\mathbf{L})^{-1}\left[(1-w)\,\mathbf{1} - w\mathbf{U}\right]$$

and

$$\mathbf{a} = \mathbf{\Lambda}^{-1}\,\mathbf{b}$$

The parameter w, to be chosen, is in the range of zero to two. For the special case $w=1$, the SOR method reduces to the familiar Gauss-Seidel method. The advantage of the SOR method is that it preserves the sparsity in A in the iteration and since A is real, symmetric, and diagonally dominant, convergence is guaranteed. The running time in each iteration is linear.

6. The I/O pads on the boundary of a chip define the coordinates of the fixed modules in the x-direction by projecting all pads to the x-axis as shown in Fig. 2. The one-dimension problem is a linear placement problem. Similarly, the fixed module for the y-direction linear placement problem is shown. The solutions of the two linear placement problems give the coordinates of the movable modules in two dimensions and constitute our initial placement.

7. The result of the initial placement gives the optimal solution in terms of our original objective function. This is because $L(\mathbf{x}, \mathbf{y})$ is a convex quadratic function, thus there exists a unique global minimum. However, we have assumed in the formulation that all modules are point modules and the slot constraint is ignored. In the real situation, modules occupy finite nonzero area and their shapes vary. While the slot constraint is not pertinent to our current problem of module placement, replacing the point modules with real modules will cause module overlaps. In the next section we specifically address the overlap problem and propose an efficient partitioning method with iterations to perturb the initial solution. The key feature is that we again resort to solving linear sparse equations. We repeat the process in a top-down hierarchy to obtain the final solution.

3. Partition and Iteration

As shown in Fig. 3 we introduce a *vertical* cut line to partition the chip into two. The location of the cut is determined by the area consideration based on the position of the modules obtained from the initial placement. We sort the modules from left to right and add up the area until roughly half of the total area is reached. The cut is then made. If the cut line coincides with the center line, we proceed to the next hierarchy by performing horizontal cuts on each half. If not, we must move the cut line to the center in order to satisfy the area requirement. This calls for a modification of the placement to both halves. To achieve this, we introduce a simple heuristic as follows.

We may assume that the cut line is to the right of the center as shown in Fig. 3 without loss of generality. All modules to the right of the center line are projected on the center line as shown. We then solve the global placement problem in the left half plane by the method outlined in the previous section. However, it should be noted that among the projected modules only those modules which lie between the cut line and the center line will be included in the set of movable modules while others will be grouped with the fixed modules. We next repeat the placement algorithm for the right half region. In this case all modules form the left half plane are projected on the center line as fixed modules. At the end of this process, modules have been perturbed from the initial placement in such a way that the center line divides the modules into two sets with each occupying half the area of the chip as required by the modules. The following comments explain why the method works well and how it can be further improved. Except for the improvements mentioned in items 3 and 4 below, we have reached the next level of hierarchy for further two-way partitioning by means of horizontal cuts. The process then repeats until we reach the bottom level and all modules are finally placed.

1. A key concept used in finding the global placement of the movable modules from the given placement of the fixed modules is convexity. It is easy to prove from the property of the modified connectivity matrix B that all movable modules are restricted to be inside the convex hull of the fixed modules on the boundary. This is also obvious from the electric network analogy. For passive resistive networks, the node voltage of any node cannot exceed the maximum source voltage in the circuit. Thus we are assured that the optimum location of the point modules in each placement problem obtained from the solution of linear equations constructed from the modified connectivity matrix remains within the given region.

2. As shown in [8], the problems of partitioning and placement are similar in structure. In particular, bipartitioning is equivalent to linear placement. Thus in our approach, the result of the initial linear placement guarantees a min-cut partitioning. This explains also the success of other placement techniques based on min-cut partitioning [9, 10]. With minimum crossing between the two sides of a cut, global routing requirements between them are reduced. This is obviously a good design strategy for any styles of layout.

3. Further improvement can be made at each level of hierarchy. Let us consider the two-way partitioning of the movable modules, i.e. the left half and the right half of the center line. Let the coordinates of the movable modules be depicted with subindices according to the two halves. Thus we partition $\mathbf{x}_1$ into $\begin{bmatrix} \mathbf{x}_{1a} \\ \mathbf{x}_{1b} \end{bmatrix}$. Eq. (5) becomes

$$\mathbf{A}_{11}\,\mathbf{x}_{1a} + \mathbf{A}_{12}\,\mathbf{x}_{1b} = \mathbf{b}_1 \tag{7a}$$

$$\mathbf{A}_{21}\,\mathbf{x}_{1a} + \mathbf{A}_{22}\,\mathbf{x}_{1b} = \mathbf{b}_2 \tag{7b}$$

where $\begin{bmatrix} \mathbf{b}_1 \\ \mathbf{b}_2 \end{bmatrix} = \mathbf{b}$ is the contribution due to the given I/O pads as in Eq. (6). Clearly,

$$\mathbf{x}_{1a} = \mathbf{A}_{11}^{-1}\left[\mathbf{b}_1 - \mathbf{A}_{12}\,\mathbf{x}_{1b}\right] \tag{8a}$$

and

$$\mathbf{x}_{1b} = \mathbf{A}_{22}^{-1}\left[\mathbf{b}_2 - \mathbf{A}_{21}\,\mathbf{x}_{1a}\right] \tag{8b}$$

We have used Eqs. (8a) and (8b) to improve the solution of $\mathbf{x}_1$ after the initial placement. We have found that even one iteration of Eq. (8) has led to considerable improvement. The iteration scheme has been implemented in our program.

4. The two-way partitioning is simple in concept and in calculation. The next obvious extension is four-way partitioning. This requires the one-step division of a given region into four pieces of equal area. A simple heuristic algorithm has been developed. To distinguish two-way partitioning from four-way partitioning, we call the former program Proud-2 and the latter Proud-4. The result of Proud-4 in comparison with Proud-2 is given in Section 6. In general, Proud-4 improves the quality by two to five percent in terms of total wire length. The running time is however roughly 50 percent longer.

4. Pseudo Code

The core of this program has only 1500 lines. There are another 1500 lines for the input and output. The program is written in C and can run on a VAX machine under UNIX or VMS, and also on an APOLLO under AEGIS. We list the pseudo code for the readers' reference.

```
PROUD_PLACEMENT:
Input:     design description, net weighting, number of hierarchies, number of inner iterations
Output:    placement result.
Algorithm:
    Read in and set up the data structure.
    Assign all modules to one internal block, and assign all I/O pads to one I/O block.
    Construct the modified connectivity matrix in the sparse form.
    For each hierarchy
    {
        Repeat number of inner iterations (repeat only once for the first hierarchy)
        {
            For each internal block
            {
                Reconstruct the modified connectivity matrix for this block.
                Calculate the right hand side of the linear equation according to the
                boundary condition determined by the projections of modules or I/O pads
                in other blocks.
                Use the SOR method to solve for relative module positions.
                Temporarily cut the block into two sub-blocks according to the cut
                method for this hierarchy. Assign the modules into separate sub-blocks
                by sorting.
            }
        }
        Fix the cut at the center of each internal block and permanently associate the
        modules to the corresponding sub-blocks, which form a new set of internal blocks
        for next hierarchy.
    }
```

5. Complexity

The key consideration for storage complexity is how we store the modified connectivity matrix **B**. Since the net list specification is usually given by the pin-net connectivity, we need to express the complexity in terms of the total number of pins. However, in practice, the number of pins is roughly proportional to the number of modules. Thus the memory space complexity in our sparse matrix formulation is $O(n)$, where n is the number of modules.

The initial placement and the iterative linear system solver are linear in terms of run time complexity. The complexity for the sorting algorithm is $O(n \log n)$ at each partitioning step. The hierarchical cut creates a binary tree which can have at most $\log n$ levels. Thus the run time complexity for our method is $O(n \log^2 n)$.

6. Experimental results

The algorithm was tested on nine real circuits. Seven of them are mid-sized designs which contain 1,000 to 4,000 modules. The other two are in the realm of high-complexity layout. One has 14,000 and the other has 26,000 modules. The run time is reported under test runs on a VAX 8650 machine (which is a six MIPs machine). We observed that the memory storage requirement is so efficient that it is less than linear mainly because of the sparse matrix structure. The run time does vary from example to example because different examples need different numbers of iterations in order to converge to the final solution, and the number of cuts is different for different-sized examples. However, in general the run time is only slightly worse than linear.

Tables 1 through 9 summarize results on nine real designs and give comparisons with TimberWolf 3.2 and the recent TWOLF 405. Our results in terms of total wire length are superior to TimberWolf 3.2 in all cases where comparisons can be made. They are comparable to that of TWOLF 405. However, in terms of running time, even on the smaller chips, Proud-2 is roughly 100 times faster.

7. Conclusion

In this paper we present a novel approach to high-complexity placement. This is the first time that a deterministic algorithm is comparable with simulated annealing in the quality of the results. However, the running time is several orders of magnitude faster. The approach is especially suitable for sea-of-gates design. The method takes advantage of the inherent sparsity of the placement problem and depends on solving linear sparse equations repeatedly. In the following we indicate some future directions in extending our present method.

(i) Combining basic cells like NANDs, NORs, etc. and macros like RAMs, PLAs, etc. in one design is believed to be the most viable approach for future VLSI design. However, it is not an easy task because of the extreme size differences in cells and macros. The difficulty can possibly be overcome by the simple min-cut step in our placement algorithm which scans through a sorted list based on the *global placement* result.

(ii) The determination of the parameter w used in the SOR method needs to be studied. A good choice of the parameter can reduce the number of iterations, and will shorten the total run time.

(iii) The iterative methods for linear systems are ideal for parallel processing. Either the block Gauss-Seidel or the block SOR method can be implemented. The general issue for parallel processing is the problem of independence. If there is strong interaction among sub-problems, the costly overhead can make the parallel algorithm inapplicable [11].

(iv) The I/O pad placement is critical to the final solution of module placement. We have looked into the problem of optimum I/O pad placement from the information of module connectivity. Since all I/Os are placed around the boundary ring of a chip, which is a special case of the one-dimensional placement problem, some form of optimal solution can be expected.

(v) Timing-driven layout depends in a critical way on module placement. Our simple placement algorithm is ideal to incorporate timing into consideration.

Acknowledgement

The research is supported by the National Science Foundation under the Grant MIP-8506901 the Joint Services Electronics Program under the Contract F49620-87-C-0041, and the Hughes Aircraft Co. We would like to acknowledge Keiko Chow, Scott Evans, Chih Lee, and Jui Tang for providing help with the implementation of Proud-2. Discussions with Michael A.B. Jackson have been useful.

References

[1] Soukup, J., "Circuit Layout," *Proc. IEEE,* Vol. 69, No. 10, pp. 1281-1304, October 1981.

[2] Hall, Kenneth M., "An r-Dimensional Quadratic Placement Algorithm," *Management Science,* Vol. 17, No. 3, pp. 219-229, November 1970.

[3] Cheng, C-K. and E. S. Kuh, "Module Placement Based on Resistive Network Optimization," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems,* Vol. CAD-3, No. 3, July 1984, pp. 218-225.

[4] Sechen, C. and A. Sangiovanni-Vincentelli, "TimberWolf3.2: A New Standard Cell Placement and Global Routing Package," in *Proc. 23rd DAC,* pp. 432-439, 1986.

[5] Sechen, C. and K-W. Lee, "An Improved Simulated Annealing Algorithm for Row-Based Placement," to appear in *Proc. ICCAD,* 1987.

[6] Polak, E. "Computational Methods in Optimization," Academic Press, pp. 7-12, 1971.

[7] Golub, G.H. and C. F. Vanloan, "Matrix Computations," Johns Hopkins University Press, pp. 353-372, 1983.

[8] Tsay, Ren-Song and E. S. Kuh, "A Unified Approach to Circuit Partitioning and Placement," *20th Proceedings of Princeton Conference on Information Sciences and Systems,* pp. 155-160, 1986.

[9] Breuer, Melvin, "Min-Cut Placement," in *J. Design Automation and Fault Tolerant Computing,* Vol. 1, No. 4, pp. 343-362, October 1977.

[10] Lauther, U. "A Min-Cut Placement Algorithm for General Cell Assemblies Based on a Graph Representation," in *Proc. 16th DAC,* pp. 1-10, 1979.

[11] Tsay, Ren-Song and K-T Cheng, "Fast Parallel Algorithms for Linear System Solvers," class report, UC Berkeley, 1985.

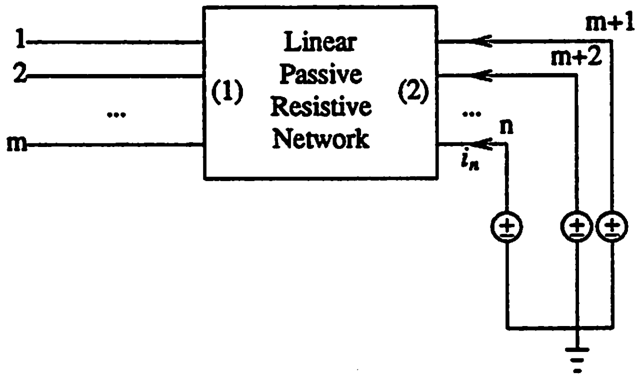

Fig. 1 An n-terminal linear, passive resistive network whose first m nodes are floating and the remaining n-m nodes are connected to voltage sources.

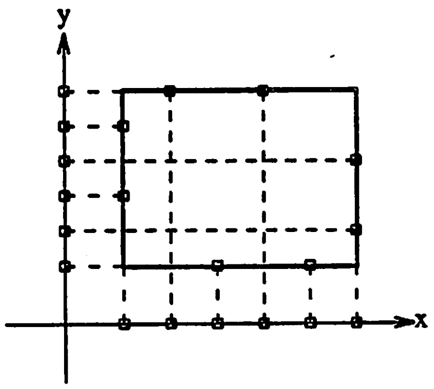

Fig. 2 Fixed I/O pads are projected to the x-axis and y-axis respectively in defining the two linear placement problems

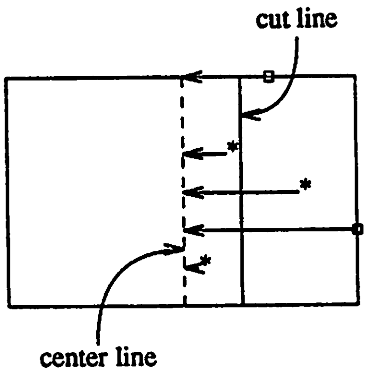

Fig. 3 Projecting modules from the right-half-plane to the center line to determine the left-half-plane placement

Table 1

design	#modules	#nets	#pins
h1	1180	1704	4815

algorithm	wire length (half perimeter)	run time (seconds)	memory (Megabytes)
PROUD-2	4025290 (1)	34 (1)	0.8
PROUD-4	3970740 (0.986)	55 (1.62)	0.8
TWOLF	NA	NA	NA
TWOLF405	3758618 (0.933)	3279 (96)	NA

Table 2

design	#modules	#nets	#pins
h2	1438	1645	4936

algorithm	wire length (half perimeter)	run time (seconds)	memory (Megabytes)
PROUD-2	3580000 (1)	50 (1)	1.27
PROUD-4	3451000 (0.964)	96 (1.93)	1.27
TWOLF	3836410 (1.072)	7200 (144)	NA
TWOLF405	3236228 (0.904)	3260 (65)	NA

Table 3

design	#modules	#nets	#pins
h3	1975	2302	9886

algorithm	wire length (half perimeter)	run time (seconds)	memory (Megabytes)
PROUD-2	5435550 (1)	232 (1)	1.4
PROUD-4	5303120 (0.976)	256 (1.10)	1.4
TWOLF	6253750 (1.151)	27973 (120)	NA
TWOLF405	4740746 (0.872)	NA	NA

Table 4

design	#modules	#nets	#pins
h4	2552	3864	12240

algorithm	wire length (half perimeter)	run time (seconds)	memory (Megabytes)
PROUD-2	14971934 (1)	86 (1)	1.63
PROUD-4	14048436 (0.938)	153 (1.78)	1.63
TWOLF	17704610 (1.183)	NA	NA
TWOLF405	14580270 (0.974)	10440 (121)	NA

Table 5

design	#modules	#nets	#pins
h5	2886	3755	11629

algorithm	wire length (half perimeter)	run time (seconds)	memory (Megabytes)
PROUD-2	7950754 (1)	99 (1)	1.56
PROUD-4	7812430 (0.983)	157 (1.59)	1.56
TWOLF	9202952 (1.157)	NA	NA
TWOLF405	NA	NA	NA

Table 6

design	#modules	#nets	#pins
h6	3249	4237	13166
algorithm	**wire length (half perimeter)**	**run time (seconds)**	**memory (Megabytes)**
PROUD-2	5328358 (1)	128 (1)	1.53
PROUD-4	5204386 (0.967)	195 (1.53)	1.53
TWOLF	8398243 (1.576)	NA	NA
TWOLF405	5008952 (0.940)	14400 (113)	NA

Table 7

design	#modules	#nets	#pins
h7	3816	3094	10778
algorithm	**wire length (half perimeter)**	**run time (seconds)**	**memory (Megabytes)**
PROUD-2	6312228 (1)	325 (1)	1.54
PROUD-4	6347820 (1.006)	654 (2.01)	1.54
TWOLF	7002087 (1.109)	NA	NA
TWOLF405	5828984 (0.923)	24108 (74)	NA

Table 8

design	#modules	#nets	#pins
h8	14091	16958	56377
algorithm	**wire length (half perimeter)**	**run time (seconds)**	**memory (Megabytes)**
PROUD-2	34518106 (1)	1704 (1)	6.12
PROUD-4	32657287 (0.946)	3143 (1.84)	6.12
TWOLF	NA	> 49 days†	NA
TWOLF405	NA	NA†	NA

† Cannot place the whole chip flat.

Table 9

design	#modules	#nets	#pins
h9	26277	29151	92530
algorithm	**wire length (half perimeter)**	**run time (seconds)**	**memory (Megabytes)**
PROUD-2	50028939 (1)	3048 (1)	11.08
PROUD-4	48142839 (0.962)	5598 (1.84)	11.08
TWOLF	NA	NA†	NA
TWOLF405	NA	NA†	NA

† Cannot place the whole chip flat.