Computer-Aided Design and Applications
© 2008 CAD Solutions, LLC
http://www.cadanda.com

Placement and Routing in Computer Aided Design of Standard Cell Arrays by Exploiting the Structure of the Interconnection Graph

Ioannis Fudos[1], Xrysovalantis Kavousianos[1], Dimitrios Markouzis[1] and Yiorgos Tsiatouhas[1]

[1]University of Ioannina, {fudos,kabousia,dimmark,tsiatouhas}@cs.uoi.gr

ABSTRACT

Standard cell placement and routing is an important open problem in current CAD VLSI research. We present a novel approach to placement and routing in standard cell arrays inspired by geometric constraint usage in traditional CAD systems. Placement is performed by an algorithm that places the standard cells in a spiral topology around the center of the cell array driven by a DFS on the interconnection graph. We provide an improvement of this technique by first detecting dense graphs in the interconnection graph and then placing cells that belong to denser graphs closer to the center of the spiral. By doing so we reduce the wirelength required for routing. Routing is performed by a variation of the maze algorithm enhanced by a set of heuristics that have been tuned to maximize performance. Finally we present a visualization tool and an experimental performance evaluation of our approach.

Keywords: CAD VLSI design, geometric constraints, graph algorithms.
DOI: 10.3722/cadaps.2008.325-337

1. INTRODUCTION

VLSI circuits become more and more complex increasingly fast. This denotes that the physical layout problem is becoming more and more cumbersome. From the early years of VLSI design, engineers have sought techniques for automated design targeted to a) decreasing the time to market and b) increasing the robustness of the final product. Traditional CAD tools for placement and routing appear to have increasing difficulty to respond adequately to modern requirements. The majority of CAD systems aim to decrease the total wirelength and subsequenlty minimize the required layout area and maximize the circuit speed [3, 9, 20]. However, using solely the criterion of minimizing the total wirelength is often not sufficient for detecting the optimal physical layout [1, 23] since such techniques do not take into account congestion. For handling connection congestion researchers have introduced annealing algorithms [2], multiple partition techniques [12], concurrent detection in top-down placement [21] and integer linear programming (ILP) [23]. Our work introduces a novel technique for placement and routing inspired by standard CAD techniques such as computational geometry algorithms [25], systems for handling geometric constraints (see for instance [4, 5, 7, 15), and graph algorithms for detecting dense structures in graphs [8]. Our approach can be used in conjunction with techniques that refine the layout progressively such as [11]. Our approach falls under the category of routing driven placement. Very efficient algorithms for routing only [16] and placement only [18] can be used for placement and routing components. Recently [17] has attempted to coordinate efficiently the routing and placement objectives by careful refinement with interesting results. All these approaches can potentially benefit form our graph analysis based method and vice-versa.

The standard-cell placement problem has drawn extensive research attention in the VLSI CAD area [24]. One common classification for traditional placement methods is: min-cut placement, simulated annealing, analytical methods, and force-directed approaches. However, recently proposed placement tools use more or less hybrid approaches. These classical techniques, plus clustering and flow-based methods, frequently appear in these relatively new placement algorithms. In addition to the above approaches that address half-perimeter wirelength, many techniques are proposed for timing and congestion optimization. Most of them are based on wirelength minimization. Wirelength is the fundamental objective in standard-cell placement problem. It is generally believed that a timing or congestion oriented approach can hardly be successful without a good wirelength minimization engine. The idea of timing driven placement is to reduce the wirelengths on certain paths instead of the total wirelength. A placement with shorter total wirelength is relatively easier to be modified to meet timing constraints. Similarly, a good placement with optimized wirelength has a higher probability that its congested regions are relatively smaller or less serious. This is the approach that we adopt in this work.

2. PRELIMINARIES

Digital circuits are interconnected sets of logical gates that perform operations varying from very simple such as AND, OR, NOR, XOR to very complex such as programmable general purpose processors, image processing, graphics processing.

For small-scale logic, designers use prefabricated logic gates from families of devices. Increasingly, these fixed-function logic gates are being replaced by programmable logic devices, which allow designers to pack a huge number of mixed logic gates into a single integrated circuit. Field-programmable gate arrays (FPGAs) consist of an array of identical programmable logic blocks (PLBs) connected through a mesh like programmable network that is based on switch matrices. A circuit design is mapped into PLBs and the actual location of each PLB in the array is determined during placement. The placement algorithm assigns locations to the PLBs aiming to reduce the critical path of the circuit and ensure that the resulted circuit can be routed. Then, routing is performed where the actual wire segments and switches used to connect the PLBs are determined [13, 14]. The field-programmable nature of *FPGAs* has removed the 'hard' property of hardware; it is now possible to change the logic design of a hardware system by reprogramming some of its components, thus allowing the features or function of a hardware implementation of a logic system to be modified.

In semiconductor design, standard cell methodology is the way of designing *Application Specific Integrated Circuits* (*ASICs*). *Standard cell* methodology is an example of design abstraction, whereby a layout view of a logic function is encapsulated into an abstract logic representation (such as a NAND gate). Cell-based methodology (the general class that standard-cell belongs to) makes it possible for one designer to focus on the high-level (logical function) aspect of digital-design, while another designer focused on the implementation (physical) aspect. Along with semiconductor manufacturing advances, standard cell methodology was responsible for allowing designers to scale ASICs from comparatively simple single-function ICs (of several thousand gates), to complex multi-million gate devices (*Systems on Chips - SoC*).

Strictly speaking, a 2-input NAND or NOR function is sufficient to form any arbitrary boolean function set. But in modern ASIC design, standard cell methodology is practiced with a sizeable library (or libraries) of cells (or cell units). The library usually contains simple as well as complex logic functions. Moreover, multiple implementations of the same logic function, differing in area and speed are provided. This variety enhances the efficiency of automated synthesis tools especially for designs with strict area, timing and power constraints. Indirectly, it also gives the designer greater freedom to perform implementation tradeoffs (area vs speed vs power consumption.) A complete group of standard cell descriptions is commonly called a technology library.

Commercially available *Electronic Design Automation* (EDA) tools use the technology libraries to automate synthesis, placement, and routing of a digital ASIC. The technology library is developed and distributed by the foundry operator. The library along with a logic level design *netlist* (list of connections) is the basis for exchanging design information between different phases of the *SPR (Standard Cell Place and Route)* process.

Using the technology library's cell logical view (also called *symbol*), the synthesis tool performs Cell Library Binding, i.e. the process of mathematically transforming the ASIC's *register-transfer level (RTL)* description into a technology-dependent netlist. (This process is analogous to a software compiler converting a high-level C-program listing into a processor-dependent, assembly language listing.) The *netlist* is the standard cell representation of the ASIC design, at the logical view level. It consists of instances of the standard-cell library units (gates, flip-flops or more complex designs), and port-connectivity among these cells. Proper synthesis techniques ensure mathematical equivalency between the synthesized netlist and original RTL description. The netlist contains no unmapped (RTL) statements and declarations.

The placer tool starts the physical implementation of the ASIC. With a 2-D floorplan provided by the ASIC-designer, the placer tool assigns locations for each standard-cell in the netlist. The resulting placement netlist contains the physical location of each of the netlist's standard-cells, but retains an abstract description of how their' terminals are wired to each other.

Typically standard cells have a constant height that allows them to be lined up in rows on the integrated circuit. The chip will consist of a very large number of rows (with power and ground running next to each row) with each row filled with the various cells making up the actual design. Placers obey certain rules: each cell is assigned a unique (exclusive) location on the diemap. A given gate is placed once, and may not occupy or overlap the location of any other gate. Using the placement netlist and the layout view of the standard cells, the router adds both signal connect lines and power supply lines. The fully routed (physical) netlist contains the listing of cells (from synthesis), the placement of each cell (from placement), and the drawn interconnects (from routing).

ASIC design strategies are usually categorized into three broad groups:

- *Full custom*, where the designer must design everything from scratch. This design strategies offers optimized performance of the final product at the cost of very high design effort and very long design time.
- *Semi custom*, where we have in our disposal a library of logic unit designs that we can combine to create more complex circuits. Even though the performance of the final product can not be optimized as much as in the full custom design strategy, both the design effort and the design time are greatly improved. Gate Arrays and Standard Cells fall under this broad category.
- *Programmable*, where chips are ready and we only need to program the function that we wish to implement. This strategy minimizes the development time and effort, at the cost of reduced performance.

Standard cell technology is a prevailing semi custom VLSI design paradigm for the following reasons:

- The designer is able to design very large designs using only the cell library provided and the appropriate CAD tool.
- The rules that apply guarantee that the risk of failing to function properly will be small.
- The design time is minimized since the designer does not bother with very low level details. On the other hand breakthroughs in the low level are made easily available to all standard cell user through new components of the cell library.

The process of VLSI design can be thought as a sequence (see Fig. 1) of the following steps: The outcome of the first step, *high level synthesis*, is usually a high level description of the circuit function and is expressed using high level language that describe the RTL that implements digital circuit behavior. The second step, *logic synthesis optimization*, performs an optimization on the result of the first step targeted to area minimization, power dissipation minimization and speed optimization. Subsequently the circuit is simulated and its operation is verified (*simulation verification*). Last but not least before manufacturing is the step of layout synthesis (placement and routing) which we examine in this work. This step is usually performed in four substeps:

1. *System partitioning*: the system circuitry is partitioned in circuits that usually perform a distinct function.
2. *Floor planning*: the chip is partitioned in “rooms” called blocks similarly to architectural design and each block is assigned to a part determined in the first step.
3. *Placement*: the placement of standard cell units inside each individual block area of the floor planning is determined.
4. *Routing*: all connections (interblock and intrablock) are routed so as to minimize the wirelength and the number of routing metal levels.

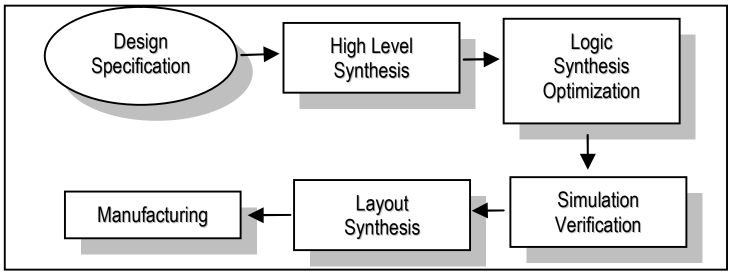
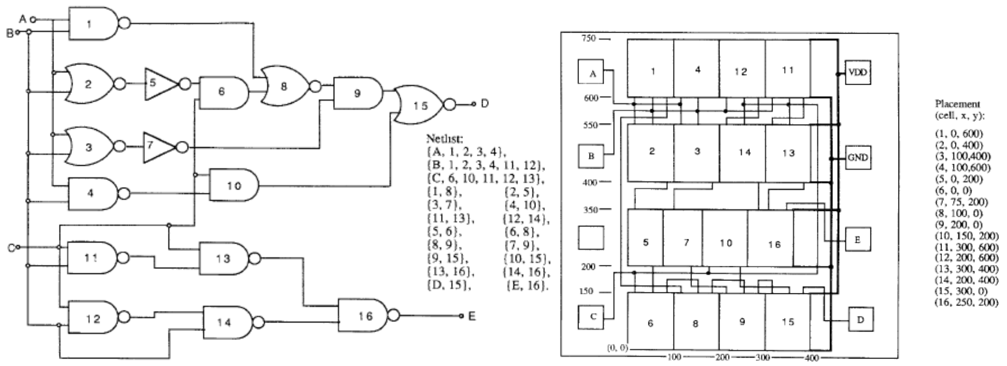


Fig. 1: Design flow for ASICs.

In particular in standard cell technology we have to deal mostly with the two last steps, i.e. placement and routing. In standard cells the constrained problem takes a more constrained form. Each row has the same height and we are supposed to place standard cells of variable width on any of the rows (see Fig. 2 - right).

Each cell has a predefined width and location of the output, input and power pins (see e.g. an AND gate implementation in Fig. 3).



```
Netlist:
{A, 1, 2, 3, 4},
{B, 1, 2, 3, 4, 11, 12},
{C, 6, 10, 11, 12, 13},
{1, 8},        {2, 5},
{3, 7},        {4, 10},
{11, 13},      {12, 14},
{5, 6},        {6, 8},
{8, 9},        {7, 9},
{9, 15},       {10, 15},
{13, 16},      {14, 16},
{D, 15},       {E, 16}.
```

```
Placement
(cell, x, y):

(1, 0, 600)
(2, 0, 400)
(3, 100,400)
(4, 100,600)
(5, 0, 200)
(6, 0, 0)
(7, 75, 200)
(8, 100, 0)
(9, 200, 0)
(10, 150, 200)
(11, 300, 600)
(12, 200, 600)
(13, 300, 400)
(14, 200, 400)
(15, 300, 0)
(16, 250, 200)
```

Fig. 2: (left) A circuit described by its netlist and its schematic diagram. (right) Placement and routing in standard cell technology.

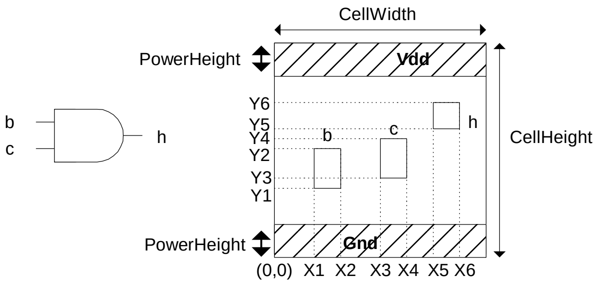


Fig. 3: Detail of a Cell.

3. CELL PLACEMENT

In this section we present a method that places the cells by traversing the induced interconnection graph of the cells using a depth first search algorithm. Then, we present an improvement that takes further into consideration the structure of the interconnection graph.

3.1 Introduction

The input in our case is the netlist and the physical layout of each cell representing the standard cell units that are being used. The *netlist* is a description of the standard cells and their interconnection. Fig. 2 (left) illustrates the netlist of a circuit along with the corresponding schematic diagram.

A placement for this circuit is shown in Fig. 2 (right). The physical implementation of each cell that is to be used is described by providing the width of the cell and the exact location of areas of its input and outputs.
There are two broad classes of placement algorithms:

1. *Constructive algorithms* select a standard cell unit place it on the cell array and then the rest of the standard cell units are placed one at a time based on their connection with the already placed ones.

2. *Iterative improvement algorithms* are based on exhaustive combinatorial optimization. A first good guess is presented and then a pair or more standard cell units are swapped and this modification is accepted based on a sophisticated cost function.

Our approach is a constructive approach enhanced by a global analysis of the interconnection graph.

3.2 A DFS Algorithm for Placement

Our basic algorithm is based on the Depth First Search (DFS) strategy for visiting and placing all standard cell units. The interconnection graph of a circuit is built from the netlist and is represented using adjacency lists. For the circuit of Fig. 4 (left) the corresponding graph is shown in Fig. 4 (right).

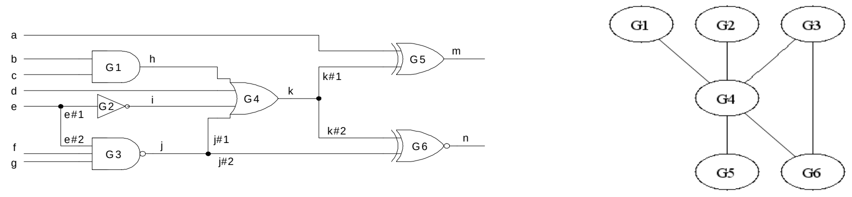


Fig. 4: (left) A small circuit and (right) the corresponding undirected interconnection graph.

We first select a node that is best suited for being placed in the center of the cell. We may use various criteria for selecting this node. We select the node that roots a minimum height spanning tree of the graph. This is called the *source node.* From this node, we initiate a DFS and visit and place all nodes of the graph around the source node in a spiral layout.

The placement is guided by the following rules that were determined experimentally:

1. Initially every node (representing a standard cell unit) visited is attempted to be placed on the same row as its DFS parent next to it either on its right or on its left. If there is no spot that can accommodate the node adjacent to its DFS parent on the same row we place it on the same column either right above or right below the parent node.
2. If there is no available adjacent (to the parent node) spot to place the new standard cell, we examine whether the row of the parent node has enough available space in total to accommodate the new node. If this is the case we shift the surrounding nodes to the left or to the right to create available space as close as possible to the parent node. Note that certain clusters of nodes in previous steps of placement have been marked as connected to avoid separating them later on. However we are still capable of moving whole groups of connected nodes to the left or to the right.
3. Step 2 is repeated for the row just above the parent row and if this returns no result for the row just below the parent row.
4. If Steps 1-3 do not return an available placement candidate position, we place the node in the free area of the cell array that is closest (using the Manhattan distance) to the parent node.

We illustrate the above process by the placement of the components of a circuit with 11 standard cell units that have the same width. The interconnection graph is shown in Fig. 5.

G1 is selected as a root of a minimum height spanning tree. Then we place G4, G5 (Fig. 6 - left - upper), G12, G2, G7, G8 (Fig. 6 – right - upper). Then G6, G3 are visited and placed (Fig. 6 – left - lower) and finally G9, G10 and G11 are visited and placed successfully (Fig. 6 – right -lower).

Note that the DFS traversal may result in different placements and that in general no assumption is being made for the width of the cells that correspond to a graph node. Fig. 7 illustrates the overall algorithm. The algorithm has time complexity $O(|E|)$, where $|E|$ is the number of edges (connections) of the graph.

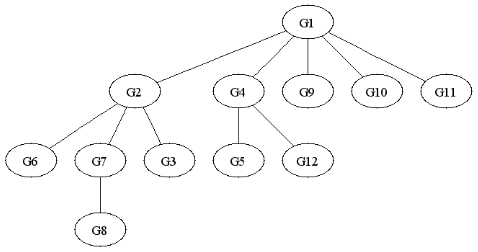


Fig. 5: An example circuit interconnection graph.

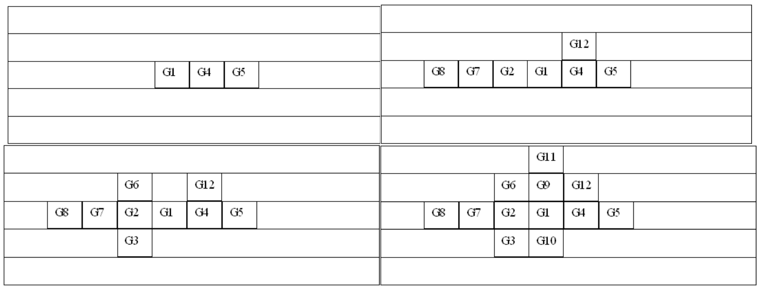


Fig. 6: Steps of DFS traversal and placement.

```
DFS (G)
    1.  for each vertex u∈G
    2.  u[visit] := FALSE
    3.  endfor
    4.  For each vertex u∈G
    5.  if u[visit] FALSE then
    6.  u[father] := NULL
    7.  DFS-VISIT(u)
    8.  endIf
    9.  endfor

DFS-VISIT(u)
    1.  u[visit] := TRUE
    2.  PLACE(u)
    3.  for each v∈Adj[u]
    4.  if v[visit] = FALSE
    5.  v[father] := u
    6.  DFS-VISIT(v)
    7.  endIf
    8.  endfor

PLACE(u)
    1.  if Area is empty then
    2.      place  u in the middle of the Area
    3.  else
    4.      if there is enough place exactly right of  u[father] then
    5.          place u right of u[father]
    6.      else if there is enough place exactly left of  u[father]
        then
    7.          place u left of  u[father]
    8.      else if there is enough place exactly above u[father]
    9.          then place u above u[father]
    10.     else if there is enough place exactly below u[father] then
    11.         place u below u[father]
    12.     else if there is enough place right of  u[father] then
    13.         move cells right of father and place u
    14.     else if there is enough place left of  u[father] then
    15.         move cells left of  father and place u
    16.     else if there is enough place above u[father] then
    17.         move cells left or right and place u exact above
                u[father]
    18.     else if there is enough place below u[father] then
    19.         move cells left or right and place u exact below to
                u[father]
    20.     else if there is a line up to u[fahter] with free space
                    exactly above  u[father]
    21.         place u
    22.     else if there is a line down to u[father] with free space
                    exactly below to u[father]
    23.         place u
    24.     else if there is free area space wherever then
    25.         place u
    26.     else
    27.         return FAILED.
    28.     endIf
    29. endIf
```

Fig. 7: The DFS traversal placement algorithm.

3.3 A Density-based Placement Variation of DFS

We present a variation that is inspired by geometric constraint solving in CAD/CAM systems. In this approach we first partition the circuit into a number of more dense circuits and then attempt to place dense circuits around the center of a conceptual spiral, thus minimizing the wirelength needed in the routing step. We analyze the graph of the system to detect dense induced subgraphs which are then reduced into single nodes representing the corresponding block. Then, instead of starting a single DFS from a random node, we place the denser graphs around the center by performing a DFS traversal on each one of them.

Let G= (V, E) be the interconnection graph. Traditional graph density (i.e. |E| - |V| > K) does not always yield beneficial partitionings. To overcome this problem we revert to a more flexible version of density that was introduced in [8] having in mind geometric constraint solving. Here G is augmented with weights, G= (V, E, w), where w is a weight function for all nodes and edges that returns positive integer values. In our application, we assume that the weight has two distinct values: one for each edge and one for each node. The graph density is then given by Eqn. (3.1).

$$d(G) = \sum_{e \in G} w(e) - \sum_{u \in G} w(u) \tag{3.1}$$

An induced subgraph A of G is called dense when its density is larger than a constant K. K, w(e) and w(v) are all characteristics that are derived from attributes of the initial graph G.

$$d(A) = \sum_{e \in A} w(e) - \sum_{u \in A} w(u) > K \tag{3.2}$$

K is an integer and may also take negative values. Assigning weights and determining K is a key issue for tuning the efficiency of the algorithm. The weights are assigned so as the initial graph is dense, this guarantees that the algorithm will return graphs more dense than the initiall graphs and that if no such subgraph exists the initial graph will be returned. For example if we set w(e) for every e, then: $w(v) = floor(\frac{|E|}{|V|} * w(e))$. For example, with w(e)= 10 for every node, |V|= 160 nodes and |E|= 255 edges we have w(v)= 15.

We use the efficient minimal dense graph detection algorithm developed in [8]. The algorithm runs in time O(|V||E|). It proceeds by building a network flow and trying to distribute the weight of the edges in this network. If this is not possible, then we have detected a dense graph. Fig. 8 illustrates the detection and reduction of a dense subgraph.

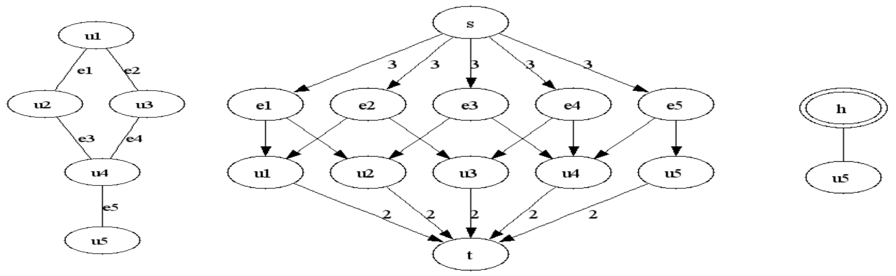


Fig. 8: (left) The initial graph. (center) detecting a dense graph using network flow. (right) Substituting the dense graph (u1, u2, u3, u4) by a hypernode h.

4. ROUTING

We present a maze algorithm that takes into account connection congestion. This algorithm developed returns a solution which is very close to the optimal solution derived without taking into account congestion. The algorithm presented here works on the placement outcome described in Section 3. This approach is built targeting to minimize: (a) the total numbers of metal routing levels used, and (b) the total wirelength.

4.1 Introduction

The routing problem is a hard combinatorial problem. Almost all real world versions of the routing problem are NP-hard. Thus, we are always trying to find approximate algorithms based on heuristics or branch and bound strategies. *Global routing* algorithms are not concerned with detailed geometric characteristics of the connections. Such approaches usually aim to minimize the total area needed for interconnection. In *detailed routing* we usually determine specific feasible connection channels and layers. In Standard Cells we have by default two metal levels, one accommodating the vertical lines and the other level the horizontal lines. If given a certain area and placement routing is not feasible with two routing layers, the number of layers may increase to 4, 6 and so on.

Many systems employ maze algorithms or Steiner-trees based algorithms. The former are greedy algorithms, which usually derive close to optimal solutions, but may become very slow when presented with a large number of input nodes (see e.g. [6, 10, 19]). The latter are fast algorithms that provide the optimal solution without considering the line congestion problem. Finally line probe algorithms are fast algorithms for detailed routing. Their basic flaw is that they may not be able to find a route between two standard cell units even if one exists. Because of this they result in an increase in the number of layers. Finally we have algorithms especially for connecting the power and the clock, these algorithms have very specific restrictions and are usually applied before any other routing scheme.

4.2 A Maze Router

The proposed algorithm is a detailed routing algorithm. For the purposes of routing we consider a grid of plausible routing positions. The grid cells size is determined by the wire width and the spacing requirements provided in the input file and supported by a given manufacturing technology (see Fig. 9). This also determines the total numbers of grid rows and columns.

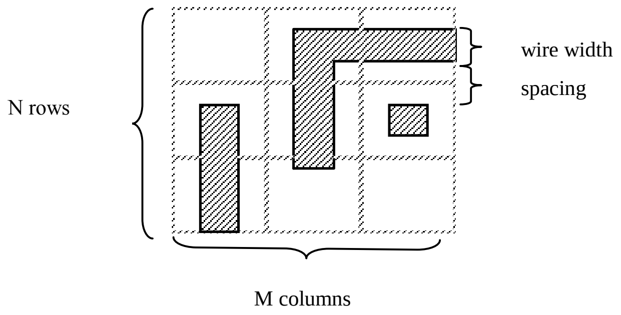


Fig. 9: The wire width and the in between spacing requirements determine the grid size.

In routing we consider at each step the connection of the output of one standard cell unit towards the input of one or several other standard cell units. The former is called *single terminal net* and the latter *multi terminal net.*

4.2.1 Single Terminal Net

Our objective is to connect two standard cells with the minimum total Manhattan distance which is identical to the minimum total wirelength. Given a pair of source and target, we seek to find a route connecting them. First we detect the grid points closest to both. Then we start from the source point of the grid and we move on the grid trying to detect the best route towards the grid point of the target. At each step, we consider a set of plausible movements in the grid (up, down, left, right). This creates a large set of plausible grid elements that we need to consider. This set is often called *wavefront.* Once the target is reached we go backwards and select the shortest route. Summarizing, we perform the following:

- a BFS (Breadth First Search) on the grid starting from the source towards the target
- select the best route going backwards, marking the corresponding cells as used.
- clear all other paths for further use.

Every grid square can be crossed both horizontally and vertically without collision (this corresponds to the first two layers). Fig. 10 illustrates an application of the single terminal router. The left table shows the source and the target, the center table shows the manhattan distances from the source at neighboring squares after two steps. In the last step (step 4) V stands for vertical wire (level 1) and H stands for horizontal wire (level 2). B is the mark for a via, i.e. a connection between level 1 and level 2. A grid square marked as H can be used later on for a vertical wire, a grid square marked as V can used later on for a horizontal wire. Finally, a grid square marked as B cannot be used later on (in levels 1 and 2).

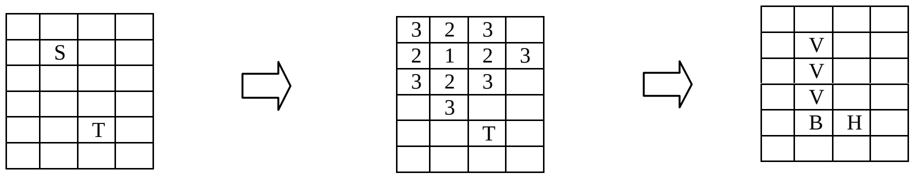


Fig. 10: A single terminal wavefront.

4.2.2 Multi Terminal Net

The algorithm is quite similar to the single terminal net maze router. Here we start from the source until we meet the closest target. Then we find the shortest path from the source to the closest target and then we mark all grid squares on this path as sources. Then we expand the wavefront from all sources concurrently until we meet the next target. Fig. 11 illustrates this process.

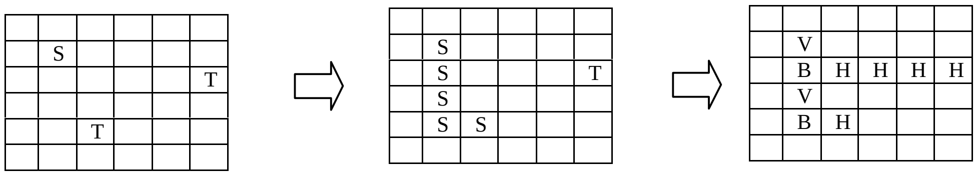


Fig. 11: Multi Terminal Routing.

4.2.3 Heuristics and Data Structures for Speed-up

To speed up the routing process in single terminal connections we define a cost for each candidate grid cell at position i, j of the grid as follows (d denotes the Manhattan distance, i.e. the L_1 Minkowski distance):

$$cellcost(i,j) = \begin{cases} d(cell_{i,j}\ ,\ target), \text{ if we are approaching target} \\ d(cell_{i,j}\ , target)\ +\ penalty, \text{ if we are going in a direction further from target} \end{cases} \quad (4.1)$$

For multi terminal connections routes we do not use a penalty term: $cell\cos t(i,j) = \sum_{k=1..t} d(cell_{i,j}, t\arg et_k), k \geq 0$. At each step we do not consider all neighbor cells but only those with the smallest cost. To make the algorithm more efficient we keep a list of candidate paths sorted according to a path cost:

$$pathcost\ = pathcost(predecessor)\ +\ 1\ +\ cellcost(\mathrm{i,j})\ +\ bend\_penalty \quad (4.2)$$

The algorithm terminates when we reach target and all cells in the wavefront have larger pathcost than the current pathcost. Fig. 12 illustrates the maze routing algorithm.

```
1.  source = FindSource (source_x, source_y)
2.  target = FindTarget(target_x, target_y)
3.  wavefront_structure = {source}
4.  while(have not hit target){
5.      if(wavefront = = empty)
6.          exit – no path found
7.      C = get lowest cost cell on wavefront_structure
8.          mark C as reached
9.      if( C = = target)
10.         if(pathcost( C ) < minimum pathcost cell on on
            wavefront_structure){
11.             backtrace path in grid
12.             cleanup
13.             return – we found a path
14.         } endif
15. } endwhile
16. for each (unreachable neighbor N of cell C){
17.     compute cost to reach it= pathcost of C +
        cellCost of C + 1 + bend_penalty
18.     mark N cell in grid with predecessor direction
        back to cell C from N
19.     add this cell N to wavefront
20. } endfor
21. delete cell C from wavefront
```

Fig. 12: An improved version of the maze routing algorithm.

5. IMPLEMENTATION AND PERFORMANCE EVALUATION

5.1 Implementation

The systems was developed in C/C++ under LINUX. We have used OpenGL and GLUT for the interface and the visualization of the resulting layout. Fig. 14 (left) illustrates a 3D visualization of the placement and routing result in two metal layers. Fig. 13 (right) illustrates a detail of two standard cell units in the circuit.

5.2 Performance Evaluation

Both the netlist and the physical layout of each cell are provided through an XNF file (Xilinx Netlist Format) [22] which is a standard for describing Logic Cell Arrays. We have performed experiments with six benchmark input XNF files.

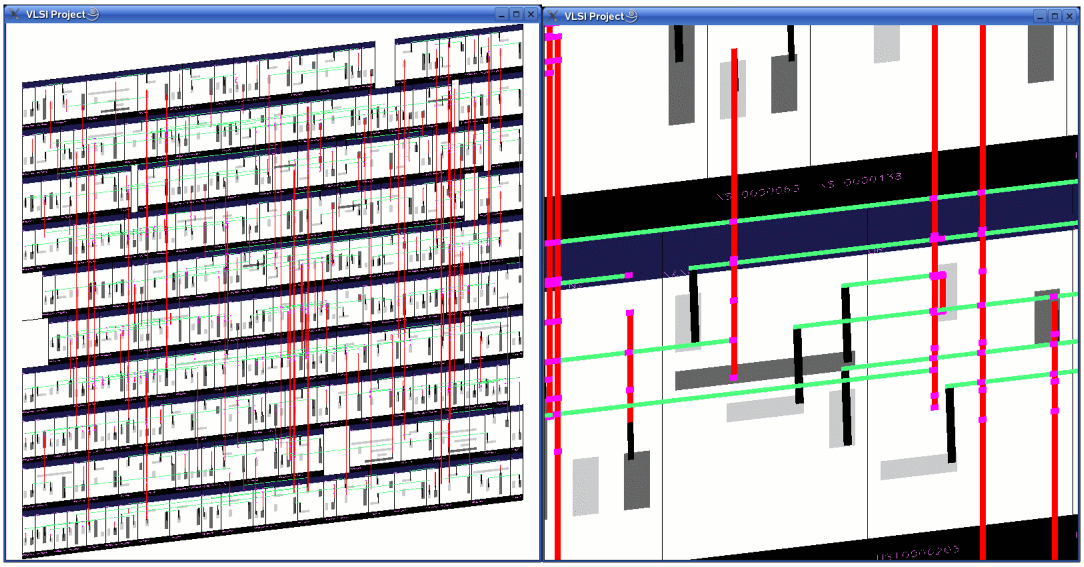


Fig. 13: (right) A 3D Visualization of a physical layout (left) Detail of two standard cells.

We have performed experiments for six input files with standard cell units from 160 up to 1389. For each input file, we perform the following experiment. For 7 different available array areas (5 in the large circuit) we perform the DFS placement algorithm and subsequently we perform the improved routing maze algorithm. We run the same experiment with the dense graph analysis algorithm and we apply the same improved maze routing algorithm. We measure:

- the overall wire length
- the time it takes to perform placement and routing (the placement time is small as compared to the routing time)
- we calculate for each placement the optimal Steiner tree based routing that does not take into account route collision and we compare its wire length with the one returned by our improved maze algorithm

Fig. 14 illustrates the wire length results for both the DFS and the dense graph based algorithms whereas Fig. 15 illustrates the total time for placement and routing for both the DFS and the dense graph based algorithms.

We observe that the algorithm that performs a dense graph analysis provides a reduced wire length from 6% up to 14% compared to the DFS algorithm. The corresponding overall time for placement and routing is reduced from 26% up to 68% if we first apply the dense graph analysis. Finally the deviation from the optimal routing result that does not consider congestion (route collisions) is very small in all cases (for both the DFS and the dense algorithm) and does not exceed 7%. This demonstrates the effectiveness of the improved maze router used.

6. SUMMARY

We have presented an improved routing algorithm for standard cells that is quite efficient and very effective. We have also presented a novel graph analysis based algorithm for placement. Finally we have performed performance evaluation based on 6 benchmark input files.

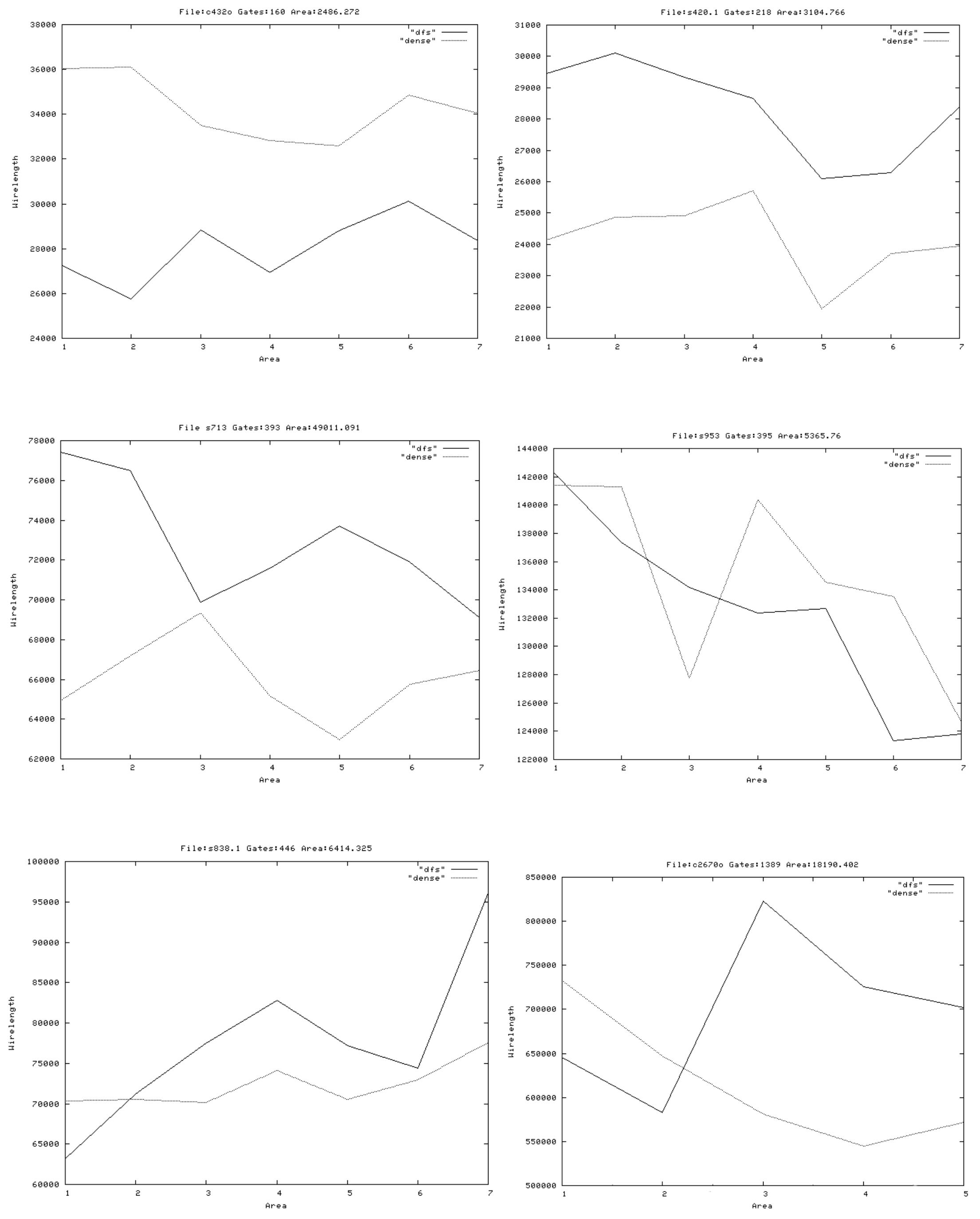


Fig. 14: Total wire length comparing the DFS and the dense graph detection technique for six input circuits.

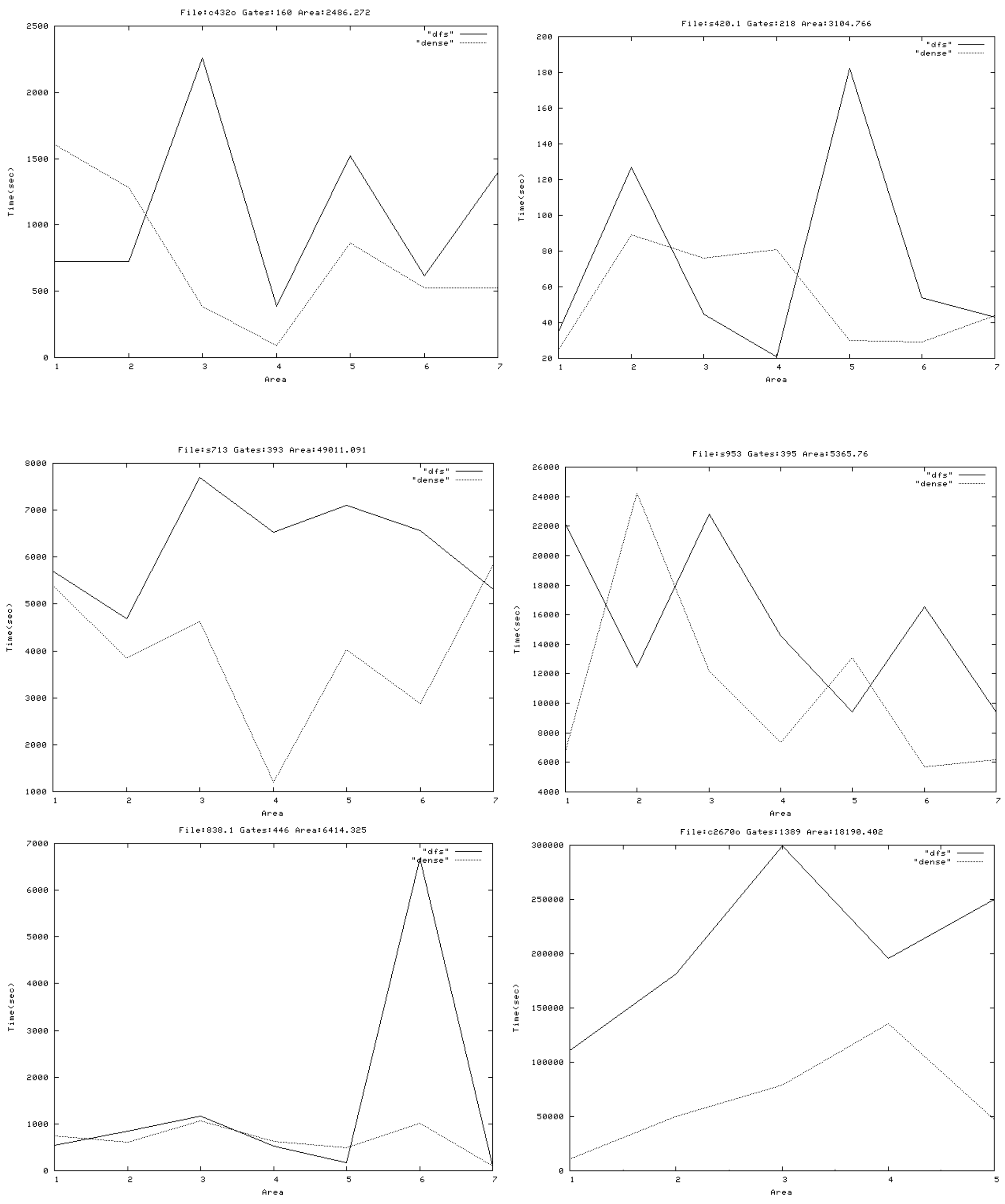


Fig. 15: Total time for placement and routing comparing the DFS and the dense graph detection technique for six input circuits.

7. REFERENCES

[1] Caldwell, A. E.; Kahng, A. B.; Markov, I. L.: Can Recursive Bisection Alone Produce Routable Placements?, In Procedings of Design Automation Conference, 2001, 477-482.

[2] Cheng, C. E.: RISA: Accurate and Efficient Placement Routability Modeling, Proceedings of the International Conference on Computer Aided Design, 1994, 690-695.

[3] Dunlop, A. E; Kernighan, B. W.: A Procedure for Placement of Standard Cell VLSI Circuits, IEEE Trans. Comput. Aided Design, 4, 1, 1985, 92-98.

[4] Fudos, I.: An Interactive Constraint Solver for Computer Aided Design, In Proceedings of EURISCON'98, June 1998, Athens, Greece.

[5] Fudos, I.; Hoffmann, C. M.: A Graph-constructive Method to Solving systems of Geometric Constraints, ACM Transactions of Graphics, 16(2), 1997, 179-216.

[6] Hightower, D.: The Lee Router Revisited, Proceedings of ICAAD, 1993, 136-139.

[7] Hoffmann, C. M.; Joan-Arinyo, R.: Symbolic Constraints in Geometric Constraint Solving. J for Symbolic Computation 23, 1997, 287-300.

[8] Hoffmann, C. M.; Lomonosov, A.; Sitharam, M.: Geometric constraint decomposition, in Geometric Constraint Solving and Appl., Bruderlin, B. and Roller, D., eds, 1998, Springer, 170-195.

[9] Kleinhans, J. M.; Sigl, G.; Johannes, F. M.; Antreich, K. J.: GORDIAN: VLSI Placement by Quadratic Programming and Slicing Optimization, IEEE Trans. Comput. Aided Design, 10, 3, 1991.

[10] Lee, C. Y.: An Algorithm for Path Connection and Its Applications, IRE Trans. Electron. Comput., Sept. 1961, 346-365.

[11] Li, C.; Xie, M.; Koh, C. K.; Cong, J.; Madden, P. H.: Routability-driven Placement and White Space Allocation, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 26(5), May 2007.

[12] Mayrhofer, S.; and Lauther, U.: Congestion-Driven Placement Using a New Multi-partitioning Heuristic, Proccedings of the International Conference on Computer Aided Design, 1990, 332-335.

[13] Mulpuri C.; Hauck, S.: Runtime and Quality Tradeoffs in FPGA Placement and Routing, International Symposium on FPGA, 2001, 29-36.

[14] Nam, G. J.; Sakallah, K.; Rutenbar, R.: A New FPGA Detailed Routing Approach via Search-Based Boolean Satisfiability, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 21, 6, 2002, 674-684.

[15] Owen, J.: Constraints on simple geometry in two and three dimensions, In Third SIAM Conference on Geometric Design, November 1993.

[16] Pan, M.; Chu, C.: FastRoute 2.0: A High-quality and Efficient Global Router, In Proceedings of Asia and South Pacific Design Automation Conference, 2007.

[17] Pan, M.; Chu, C.: IPR: An Integrated Placement and Routing Algorithm, In Proceedings of DAC 2007, San Diego, California, June 2007.

[18] Pan, M.; Viswanathan, N.; Chu, C.: An Efficient and Effective Detailed Placement Algorithm, In Proceedings of IEEE/ACM International Conference on Computer-Aided Design, 2005, 44-85.

[19] Rubin, F.: The Lee Path Connection Algorithm, IEEE Trans. on Computers, 23, 9, Sept 1974, 907-914.

[20] Sun, W. J.; Sechen, C.: Efficient and Effective Placement for Very Large Circuits, IEEE Trans. Comput. Aided Design, 14, 3, 1995, 349-359.

[21] Wang, M.; Yang, X.; Sarrafzadeh, M.: Congestion Minimization During Top-Down Placement, IEEE Trans. Comput. Aided Design, 19, 10, 2000, 1140-1148.

[22] Xilinx Inc.: Xilinx Netlist Format Specification, ftp://ftp.xilinx.com/pub/documentation/xactstep6/xnfspec.pdf

[23] Yang, X.; Wang, M.; Kastner, R.; Ghiasi, S.; Sarrafzadeh, M.: Congestion Reduction During Placement with Provably Good Approximation Bound, ACM Trans. On Design Automation of Electronic Systems, 8, 3, 2003, 316-333.

[24] Yang, X.; Choi, B. K.; Sarrafzadeh, M.: A Standard-Cell Placement Tool for Designs with High Row Utilization, In Proceedings of ICCD 2002, Freiburg, Germany, 45-46.

[25] Yao, F. F.: Computational Geometry, In Handbook of Theoretical Computer Science, ed. J. van Leeuwen, The MIT Press, 1994.