# CSEP 521 – Applied Algorithms – Winter 2007

Project: **Algorithms for VLSI Physical Design Automation**

### Pavan Adharapurapu

### March 12, 2007

# Introduction

The information revolution that has transformed our lives is driven by a revolution in Integrated Circuit (IC) technology. IC technology has evolved in the 1960s from the integration of a few transistors to the integration of millions of transistors in *Very Large Scale Integration* (VLSI) chips currently in use. The rapid growth in integration technology has been and continues to be made possible by the automation of various steps involved in the design and fabrication of VLSI chips.

The VLSI design cycle for the production of a chip starts with a formal specification of a VLSI chip, follows a series of steps, and eventually produces a packaged chip. A typical design cycle may be represented by the flowchart in Figure 1. Our emphasis is on the physical design step of this cycle.

ICs consist of a number of electronic components, built by layering several different materials in a well-defined fashion on a silicon base called a *wafer*. The designer of an IC transforms a circuit description into a geometric description called the *layout*. A layout consists of a set of set of planar geometric shapes in several layers. The process of converting the specification of an electrical circuit into a layout is called the *physical design process*.

**Definition:** VLSI Physical Design (PD) Automation is essentially the research, development and productization of algorithms and data structures related to physical design process.

The objective is to investigate optimal arrangements of devices on a plane (or in three dimensions) and efficient interconnection schemas between these devices to satisfy certain topological, geometric, timing and power-consumption constraints.

The PD process manipulates very simple geometric objects, such as polygons and lines. As a result, PD algorithms tend to be very intuitive in nature, and have significant overlap with graph algorithms and combinatorial optimization algorithms. In view of this observation, many consider PD automation the same as the study of graph theoretic and combinatorial algorithms for manipulation of objects in two and three dimensions. The PD process itself is made of multiple sub-processes including: logic partitioning, floorplanning and placement, routing (global routing and detailed routing), and compaction. This is illustrated in Figure 2. In what follows, we explain the problems involved in each of these steps and discuss the applications of a number of important optimization techniques, such as network flow, Steiner tree, scheduling, simulated annealing, generic algorithm, and linear/convex programming in solving them.

|  |  |
| --- | --- |
| Figure : VLSI Design Cycle | Figure : Physical Design Cycle |

# Classes of Graphs in Physical Design

The basic objects in PD are rectangles and lines. The rectangles are used to represent circuit block tiles and vacant tiles in a layout design. This is shown in Figure 3. The lines represent the interconnection wires. The relationship between these objects, such as overlap and distances, are very critical in development of PD algorithms. Graphs are a well developed tool used to study relationships between objects. Naturally, graphs are used to model many VLSI physical design problems and they play a pivotal role in almost all VLSI design algorithms.

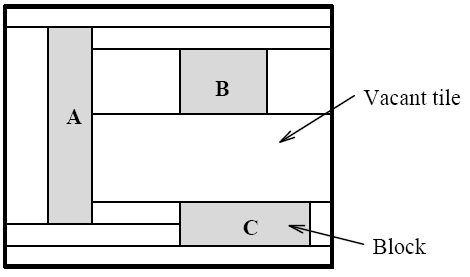


Figure : Layout partitioned into vacant and block tiles.

Many kinds of graphs are used to model PD problems. The three kinds of graphs related to a set of lines are (1) *overlap graph*: The graph has one node per line and an edge between two nodes if the corresponding lines overlap (without containment) (2) *containment graph*: There is an edge between two nodes if one line completely contains the other and (3) *interval graph*: There is an edge between two nodes if there is an overlap or containment between the corresponding lines. The graph used in relation with blocks is a *neighborhood graph* where two nodes (corresponding to two blocks) are connect if the corresponding blocks are adjacent.

Most of the optimization problems in PD are NP-hard. Even for the few problems that have polynomial time solutions, even quadratic time algorithms are sometimes infeasible because of the large number of components (> 106) involved. Another issue is the constants in the time complexity of the algorithms. In PD, the key idea is to develop practical algorithms, not just polynomial time complexity algorithms.

# Partitioning

This is the first phase of PD. A chip may contain several million transistors. Due to the limitations of memory space and computation power available, it may not be possible to layout the entire chip in the same step. Therefore, the chip is normally partitioned into sub-chips (called blocks). Figure 2(a) shows a chips partitioned into three blocks. Each block has *terminals* located at the periphery that are used to connect the blocks. The connection is specified by a *netlist*, which is a collection of *nets*. The net is a set of terminals which have to be made electrically equivalent.

A VLSI system is partitioned into several levels due to its complexity. At the highest level, a system is divided into a set of PCBs. This is called *system level* partitioning. The partitioning of a PCB into chips is called *board level* partitioning while the partitioning of a chip into smaller sub-circuits is called *chip level* partitioning.

**Problem Formulation:** The partitioning problem can be expressed more naturally in graph theoretic terms. A hypergraph G = (V,E) representing a partitioning problem is constructed with each vertex *vi* V representing a component and each *hyperedge* *ei* E joining the vertices whose components need to be connected. Thus each hyperedge is a subset of the vertex set. Normally, the hyperedge is modeled as a clique or a spanning tree over the vertex set it connects. The area of each component is denoted by . The partitioning problems is to partition V into , where

Partition is also referred to as a *cut*. The cost of partition is called the *cutsize*, which is the number of hyper edges crossing the cut. The partitioning at all levels has to deal with one or more of the following parameters: (1) Minimizing the interconnection between the partitions, (2) Minimizing the maximum delay due to partition due to the critical path crossing many partitions while satisfying the constraints for (1) Maximum number of terminals allowed, (2) Maximum area for each partition allowed and (3) The number of partitions allowed.

The min cut problem is NP-complete. Hence, we use heuristic algorithms to solve the portioning problem in practice.

**K-L algorithm:** We now discuss a well known partitioning algorithm by Kernighan and Lin who published it in their paper titled “An efficient heuristic procedure for partitioning graphs” [2]. This algorithm (called the K-L algorithm) is a bisectioning algorithm and divides the graph into two equal-sized partitions. (By applying it a number of time, we can get as many partitions as we want. ) The objective is to minimize the cut size (because it is heuristic, it is not guaranteed).

The K-L algorithm is an iterative improvement algorithm. The algorithm starts with some initial partition. Local changes are applied to minimize the cut size. Figure 5 shows an illustration of the K-L algorithm. The initial partitions are: A = {1,2,3,4} and B={5,6,7,8}. Note that the initial cutsize is 9. The next step of K-L algorithm is to choose a pair of vertices whose exchange results in the largest decrease of the cutsize *or* results in the smallest increase, if no decrease is possible. The amount by which the cutsize decreases, if vertex *vi* changes over to the other partition, is represented by

Where *inedge(i)* (*outedge(i)*) is the number of edges of vertex *i* that do not (do) cross the bisection boundary. If two vertices *vi* and *vj* are exchanged, the decrease in cutsize is *D(i) + D(j)*. In the example, a suitable vertex pair is (3,5) which decreases the cutsize by 3. A tentative exchange of this pair is made and the two vertices are then locked. This lock on the vertices prohibits them from taking part in any further tentative exchanges. The above procedure is then applied to the new partition and so on until all vertices are locked. During the process, a log of all tentative exchanges and the resulting cutsizes is stored. Figure (2) also shows this log table Note that the partial sum of the cutsize decrease *g(i)* over the exchanges of first *i* vertex pairs is given in the table. The value of *k* for which *g(k)* gives the maximum value of all *g(i)* is determined from the table. The first *k* pairs are actually exchanged. This completes an iteration and a new iteration starts. If no decrease in cutsize is possible, the algorithm terminates.

The time complexity of the K-L algorithm is O(n3). This complexity is considered too high, even for moderately sized problems. The K-L algorithm is, however, quite robust. It can accommodate additional constraints, such as a group of vertices requiring to be in a specified partition.

|  |  |
| --- | --- |
| Figure : The K-L algorithm. | Figure : Illustration of the K-L algorithm. |

**Other algorithm:** Other algorithms have been proposed for partitioning which either extend the K-L algorithm or use a completely different approach. Schweikert and Kernighan proposed the use of a net model so that the algorithm can handle hypergraphs. Fiduccia and Mattheyes reduced time complexity of K-L algorithm to *O(t)*, where *t* is the number of terminals. Two entirely different techniques – the *Simulated Annealing* and *Simulated Evolution* – have been used widely for the partitioning problem. Simulated Annealing is a special class of randomized local search algorithms. Simulated Evolution is in a class of iterative probabilistic methods for combinatorial optimization that exploits an analogy between biological evolution and combinatorial optimization.

# Floorplanning and Placement

Partitioning leads to blocks with well-defined areas and shapes (fixed blocks), blocks with approximate areas and no particular shape, a netlist specifying the connections between the blocks. The objectives of the floorplanning and placement phase is to find location of all blocks, the shapes of the flexible blocks and the pin locations for all the blocks.

Floorplanning and placement can have a big effect on the total length of the wiring used and the routing. This is illustrated in Figure 6.

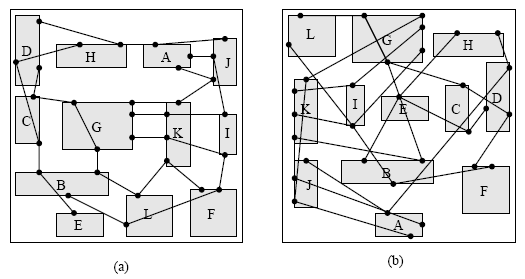


Figure : Consequences of placement.

**Problem formulation:** Given the inputs described above, the objective should satisfy the following restrictions. (1) No two rectangles overlap (2) Placement is routable (3) The total area of the rectangle bounding rectangles and net routing is minimized (4) The total wirelength is minimized.

**Algorithms used:** The general placement problem is NP-complete. Hence, various heuristic algorithms are used to solve the placement problem. These include:

1. Simulation based algorithms   
   (a) simulated annealing, (b) simulated evolution, (c) force directed placement
2. Partitioning based algorithms   
   (a) Breuer’s Algorithm, (b) Terminal Propagation Algorithm
3. Other Placement Algorithms   
   (a) Cluster Growth, (b) Quadratic Assignment, (c) Resistive Network Optimization

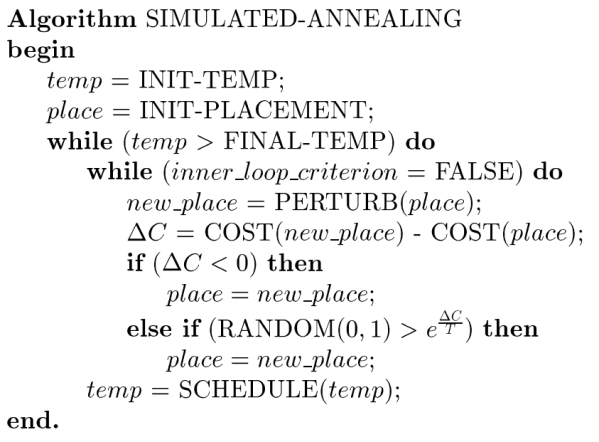
We will discuss the Simulated Annealing algorithm since it is one of the widely used algorithms. It starts with a random initial configuration and generates a new configuration by exchanging some elements. The resulting change in score is calculated. If it is negative, the move is accepted. If not, it is accepted with probability . This allows Simulated Annealing to climb out of local optimums. The Simulated Annealing algorithm is presented in Figure 7. In the context of the Placement problem, a move corresponds to any of (1) the displacement of a block to a new location, (2) the interchange of locations between two blocks, (3) an orientation change for a block.

Figure : Simulated Annealing.

# Routing

In the placement phase, the exact locations of circuit blocks and pins are determined. A netlist is also generated which specified the required interconnections. Space not occupied by the blocks can be viewed as a collection of regions. These regions are used for routing and are called as *routing regions*. Routing itself consists of two phases – Global routing and detailed routing. Figure 8 illustrates the difference between these two phases. Global routing generates a loose route for each net whereas detailed routing finds the actual geometric layout of each net. We will only discuss Global routing to give a flavor of routing in general.

|  |  |
| --- | --- |
| Figure : Routing. | Figure : Grid graph. |

Grid graph is one of the graph models used for solving the routing problem. In this, each cell is represented by a vertex. Two vertices are joined by an edge if they are adjacent to each other. The occupied cells are represented as filled circles whereas the other cells are represented as clear circles. This is illustrated in Figure 9.

**Problem formulation:** Global routing of multi-terminal nets can be formulated as a Steiner tree problem. Given a netlist and the routing graph, find a Steiner tree for each net.

Note that the problem of finding a Steiner is NP-complete. Hence, we use approximate algorithms to get an acceptable solution.

**Lee’s algorithm:** This algorithm, which was developed by Lee in 1961, is the most widely used algorithm for finding a path between any two vertices on a planar rectangular grid. The key to the popularity of Lee’s maze router is its simplicity and its guarantee of finding an optimal solution if one exists.

Lee’s algorithm uses the Grid graph described above. The exploration phase of Lee’s algorithm is an improved version of the breadth-first search. The search can be visualized as a wave propagating from the source. The source is labeled ‘0’ and the wavefront propagates to all the unblocked vertices adjacent to the source. Every unblocked vertex adjacent to the vertex labeled ‘0’ is labeled ‘1’ and so on. This process continues until the target vertex is reached or no further expansion is possible. The time and space complexity of Lee’s algorithm is *O(h\*w)* for a grid of dimension *h\*w*.

**Other algorithms:** Many other algorithms like Line-probe algorithms, shortest path based algorithms and Steiner tree based algorithms exist for solving this problem. They involve different approximations of solving the original Steiner tree problem.

# Compaction

After completion of detailed routing, the layout is functionally complete. At this stage, the layout is ready to be used to fabricate a chip. However, due to non-optimality of placement and routing algorithms, some vacant space is present in the layout. In order to minimize the cost, improve performance and yield, layouts are reduced in size by removing the vacant space without altering the functionality of the circuit. This operation of layout area minimization is called *layout compaction.* Compaction is a very complex phase in physical design cycle. It requires understanding of many details of the fabrication process such as the design rules.

**Problem formulation:** The layout of a VLSI circuit consists of geometric feature (mostly of rectangular shape). The compaction problem can be stated as: Given a set of geometric features representing a layout and the minimum size of each component and also the minimum distance between each of pair of components, the goal is to reduce the size of the components and to move the components towards each other while preserving the minimum sizes/distances.

**1-D compaction algorithm:** Many algorithms exist for solving the compaction problem. We present a *one-dimensional* (1-D) compaction algorithm, called the Shadow propagation algorithm. In 1-D compaction, components are moved in only the *x* or *y* direction. As a result, either the *x* or the *y* coordinate of the components is changed due to compaction.

An important kind of graph used in compaction graphs is called the *constraint graph.* In a constraint graph, the connections and separation rules are described using linear inequalities which can be modeled using a weighted directed graph. There exist many algorithms to generate the constraint graph itself. Well known ones include Shadow-Propagation algorithm, Scanline Algorithm, etc.

After the generation of constraint graph, the next step is to determine the critical path though the graph. The goal of one dimensional compaction is to generate a minimum width layout. The determination of minimum width layout translates into a longest path problem. The longest path from source to a vertex is then the coordinate of the vertex. The longest path problem can be viewed as a shortest path problem by inverting the signs on the edge weights. As a result, this problem is also called the critical path problem. The edges that determine the minimum distance between the source and the sink form the *critical path* and vertices on the critical path are said to be critical. A variety of algorithms exist to solve the longest path problem. These algorithms calculate for all the vertices coordinates that are as small as possible. The worst case complexity of these algorithms is *O(|V| \* |E|)*, where V is the set of vertices and E is the set of edges.

# Summary

Physical design is one of important steps in the fabrication of VLSI chips. Physical design is further divided into partitioning, placement, routing and compaction. Each of these steps is filled with innumerable graph theoretic problems, most of which are NP-complete. Researchers have come up with various heuristic, approximate and probabilistic algorithms to obtain practical solutions to these hard problems.

# Open/Research Problems

Leading researcher in the field of VLSI physical design have published the list of top 10 problems in this field. Some of them that are more algorithmic research in their nature are listed below:

1. Partitioning: One of the problems with partitioning is that it is generally treated as pure formulation. Thus, even if a terrific algorithm comes along that produces great min-cuts, a CAD developer may be skeptical of its ability to improve say, a placement algorithm. However, measuring the true effectiveness of a partitioning technique in terms of placement quality is very difficult. We need a methodology to determine the effects of one phase of PD on the succeeding phases.
2. Multi-Layer General-Area Gridless Routing: Multi-layer routing is widely used in modern IC designs. Routing is no longer confined to rectangular channels or switchboxes. A detailed routing problem is: Given a netlist, pin locations, width and spacing for each net (or wire segment), and an available multi-layer general-area routing region, generate a design-rule correct detailed routing solution with the maximum routability and the minimum total routing area.
3. Routing with incomplete data and evolving data: Given a routing region with netlist, obstacles, already routed nets, pin locations for un-routed nets, allowable layers for routing per net, width of a net. And a number stating how many % nets are missing from the netlist (say 20%), produce a routing of nets in the netlist while leaving space for un-listed nets.

# References

1. Algorithms for VLSI Physical Design Automation, 3rd edition by Naveed Sherwani. *Kluwer Academic Publishers*. All images in this paper are from this book.
2. W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49:291 – 307, 1970.
3. VLSI Physical Design Top-10 Research Problem list, maintained by leading researchers in the field at <http://www.cs.virginia.edu/pd_top10/>.