This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807

Chip Design

S. Held, S. Hougardy, J. Vygen

Research Institute for Discrete Mathematics, University of Bonn, Lennéstr. 2, 53113 Bonn, Germany

1 Introduction

An *integrated circuit* or *chip* contains a collection of electronic circuits — composed of *transistors* — that are connected by wires to fulfill some desired functionality. The first integrated circuit was built in 1958 by Jack Kilby. It contained one single transistor. As predicted by Gordon Moore in 1965, the number of transistors per chip doubles roughly every two years. Soon the process of creating chips became known as *very large-scale integration* (VLSI). In 2014, the most complex chips contain billions of transistors on a few cm^2.

In this article, we concentrate on the design of digital logic chips. Analog integrated circuits have much fewer transistors and more complex design rules, and thus are still largely designed manually. In a memory chip, the transistors are packed in a very regular structure, which makes their design rather easy. In contrast, the design of VLSI digital logic chips is impossible without advanced mathematics.

New technological challenges, exponentially fast increasing instance sizes, and shifting objectives like power consumption or yield constantly create new and challenging mathematical problems. This has made chip design one of the most interesting application areas for mathematics during the last 40 years. This will continue for at least the next two decades, although technology scaling might slow down at some point.

1.1 Hierarchical Chip Design

Due to its enormous complexity, the design of VLSI chips is usually done hierarchically. A hierarchical design makes it possible to distribute the design task to different teams. Moreover, it can reduce the overall effort, and it makes the design process more predictable and more manageable.

For hierarchical design, a chip is subdivided into logical units, each of which may be subdivided into several levels of smaller units. An obvious advantage of hierarchical design is that components that are used multiple times need to be designed only once. In particular, almost all chips are designed based on a *library* of so-called *books*, pre-designed integrated circuits that realize simple logical functions such as AND or NOT or a simple memory element. A chip often contains many instances of the same book; these instances are often called *circuits*.

The books are composed of relatively few transistors and are pre-designed at an early stage. For their design one needs to work at the *transistor level* and hence follow more complicated rules. Once a book (or any hierarchical unit) is designed, its properties that are relevant for the design of the next higher level (e.g., minimum distance constraints, timing behavior, power consumption) are computed and stored. Most books are designed so that they have a rectangular shape and the same height. This makes it easier to place them in rows or columns.

1.2 The Chip Design Flow

The first step in chip design is the specification of the desired functionality and the technology that shall be used. In *logic design*, this functionality is made precise using some hardware description language. This hardware description is converted into a *netlist* that specifies which circuits have to be used and how they have to be connected to achieve the required functionality.

The *physical design* step takes this netlist as input and outputs the physical location of each circuit and each wire on the chip. It will also change the netlist (in a logically equivalent way) in order to meet timing constraints.

Before fabricating the chip (or fixing a hierarchical unit for later use on the next higher level), *physical verification* verifies that the physical layout meets all constraints and implements the desired functionality, and *timing analysis* checks that all signals arrive in time. Further testing will be done with the hardware once a chip is manufactured.

From a mathematical point of view, physical design is the most interesting part of chip design as it requires the solution of several different challenging mathematical problems. Therefore we will describe this in more detail below.

This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807

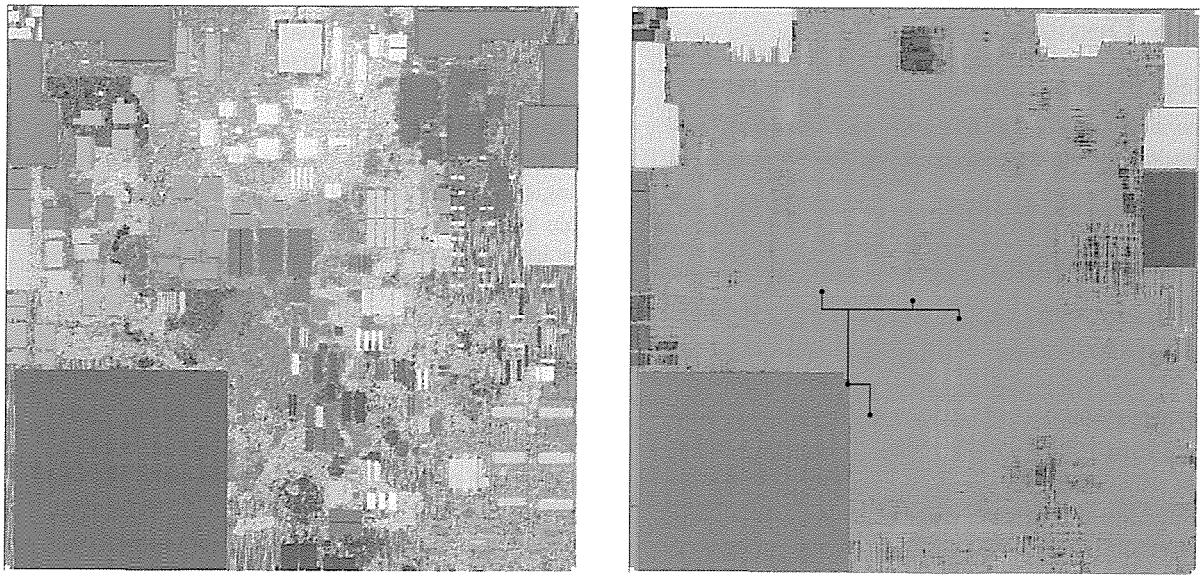

Figure 1: Placement (left) and routing (right) of a chip with 4 496 492 circuits and 5 091 819 nets with 762 meters of wires. Large rectangles are pre-designed units, e.g. memory arrays or microprocessors. On the right-hand side, a Steiner tree connecting the five pins of one net is highlighted.

1.3 Physical Design

Input of physical design is a netlist and a *chip area*. The netlist contains a set of circuits. Each circuit is an instance of a book and has some *pins* that must be connected to some other pins. Moreover, the netlist includes pins on the chip area which are called I/O-ports and connect the chip to the outside. The set of all pins is partitioned into *nets*. All pins that belong to the same net have to be connected to each other by wires.

The task of the physical design step is to assign a location to each circuit on the chip area (*placement*) and to specify locations for all the wires that are needed to realize the netlist (*routing*). Placement and routing are also called *layout* (see Figure 1).

A layout has to satisfy many constraints. For example, *design rules* specify the minimum width of a wire, the minimum distance between two different wires, or legal positions for the circuits.

Moreover, a chip works correctly (at the desired clock frequency) only if all signals arrive in time (neither too early nor too late). This is described by *timing constraints*. Usually it is impossible to meet all timing constraints without changing the netlist. This is called *timing optimization*. Of course, any changes must ensure that the netlist remains logically equivalent.

Due to the complexity of the physical design problem, placement, routing, and timing optimization are treated largely as independent subproblems. However, they are of course not independent. Placement must already ensure that a feasible routing can be found and timing constraints can be met. Changes in timing optimization must be reflected by placement and routing. Finally, routing must also consider timing constraints.

We describe some of the mathematical aspects of these three subproblems in the following.

2 Placement

Placement asks for feasible locations of all circuits on the chip area such that a given objective function is minimized. Here, a feasible location means that all circuits must completely lie within the chip area and no two circuits overlap. Normally, the chip area and all circuits have a rectangular shape. Thus, finding a feasible placement is equivalent to placing a set of small rectangles disjointly within some larger rectangle. This is

This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807

known as the *rectangle packing problem.*

No efficient algorithm is known that is guaranteed to solve the rectangle packing problem for all possible instances. However, finding an arbitrary feasible placement is usually easy in practice.

2.1 Netlength Minimization

The location of the circuits on the chip area is primarily responsible for the total wire length that is needed for wiring all the nets in the netlist. If the total wire length is too long, a chip will not be routable. Moreover, the length of the wires greatly impacts the signal delays and the power consumption of a chip. Thus, a reasonable objective function of the placement problem is to minimize the total wire length.

As this cannot be computed efficiently, estimates are used. Most notably, the *bounding box length* of a net is obtained by taking half the perimeter of a smallest axis-parallel rectangle that contains all its pins. A commonly used *quadratic netlength* estimate is obtained by summing up the squared euclidean distances between each pair of pins in the net and dividing this value by one less than the number of pins.

No efficient algorithms are known for finding a placement that minimizes the total netlength with respect to any such estimate, even if we only ask for a solution that is by an arbitrarily large constant factor worse than an optimum solution. Under additional assumptions such as all circuits having exactly the same size, one can find a placement in polynomial time whose total netlength is $O(\log n)$ worse than an optimum placement, where n denotes the number of circuits.

2.2 Placement in Practice

As mentioned above, finding an arbitrary feasible placement is usually easy. Moreover, one can define local changes to a feasible placement that results in another feasible placement. Thus general local search based heuristics (such as *simulated annealing*) can be applied. However, such methods are prohibitively slow for today's instance sizes with several million circuits.

Another paradigm, motivated by some theoretical work, is called *min-cut.* Here the netlist is partitioned into two parts, each with roughly half of the circuits, such that as few nets as possible cross the cut. The two parts will be placed on the left and right part of the chip area, respectively, and partitioned further recursively. Unfortunately, the bipartitioning problem cannot be solved well, and the overall paradigm lacks stability properties and displays inferior quality of results.

A third paradigm, *analytical placement*, is predominantly used in practice today. It begins by ignoring the constraint that circuits must not overlap; then minimizing netlength (bounding box or quadratic) is relatively easy. For several reasons (faster to solve, more stable, better spreading), quadratic netlength is minimized in practice. This is equivalent to solving a system of linear equations with sparse positive definite matrix.

The placement that minimizes quadratic netlength typically has many overlapping circuits. Two strategies for working towards a feasible placement exist: either the objective function is modified in order to pull circuits away from overloaded regions, or a geometric partitioning is done. For geometric partitioning one can assign the circuits efficiently to four quadrants (or more than four regions) such that no region contains more circuits than fit into it and the total (linear or quadratic) movement is minimized. The assignment to the regions can then be translated into a modified quadratic optimization problem.

Both strategies (as well as min-cut placement) are iterated until the placement is close to legal; that means roughly: there exists a legal placement in which all circuits are placed nearby. This ends the *global placement* phase.

After global placement (whether analytic or min-cut), the solution must be *legalized.* Here, given an illegal placement as input, we ask for a legal placement that differs from the input as little as possible. The common measure is the sum of the squared distances. Unfortunately, only special cases of this problem can be solved optimally fast enough, even when all circuits have the same height and are to be arranged in rows.

This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807

3 Routing

In routing we must connect for each net the set of its pins by wires. The positions of the pins are determined by the placement. Wires can run on different wiring planes (sometimes more than 10), which are separated by insulating material. Wires of adjacent planes can be connected by so-called *vias*. In almost all current technologies, all wire segments run horizontally or vertically. For efficient packing, every plane is used predominantly in one direction; horizontal and vertical planes alternate.

Wires can have different width, complicated spacing requirements, and other rules to obey. Although important, such rules do not change the overall nature of the problem.

Before all nets are routed, some areas are already used by power supply or clock grids. They must also be designed, but this is still largely done manually.

3.1 Steiner trees

The minimal connections for a net can be modeled as Steiner trees. A *Steiner tree* for a given set of terminals (pins) is a minimal connected graph containing these terminals and possibly other vertices; see Figure 1. If wiring is restricted to pre-defined *routing tracks*, the space available for routing a single net can be modeled as an undirected graph. Finding a shortest Steiner tree for a given set of terminals in a graph is NP-hard, and the same holds even for shortest rectilinear Steiner trees in the plane. Moreover, shortest is not always good for meeting timing constraints, and the routing graph is huge (it can have more than 10^{11} vertices). Therefore, routing algorithms mostly use fast variants of Dijkstra's algorithm in order to find a shortest path between two components, and compose the Steiner trees of such paths. If done carefully, this is at most a factor $2(t-1)/t$ worse than optimal, where t is the number of terminals (=pins in the net).

3.2 Packing Steiner Trees

Since already finding one shortest Steiner tree is hard, it is not surprising that finding vertex-disjoint Steiner trees in a given graph is even harder. In fact, it is NP-hard even if every net has only two pins and the graph is a planar grid. Nevertheless, it could be possible to solve such problems if the instance sizes are not too large.

Current detailed routing algorithms route the nets essentially sequentially, revising earlier decisions as necessary (*rip-up and re-route*). To speed up the sequential routing approach and to improve the quality of results, a *global routing* step is done in the beginning. Here, the routing space is modeled by a coarser graph, whose vertices normally correspond to rectangular areas (induced by a grid) on a certain plane. Two vertices are connected if they correspond to the same area on adjacent planes or to horizontally or vertically (depending on the routing direction of the plane) adjacent areas on the same plane. Edges have capacities, depending on how many wires we can pack between the corresponding areas.

Then global routing asks for finding a Steiner tree for each net, such that the number (more generally: total width) of Steiner trees using an edge does not exceed its capacity. This problem is still NP-hard, and the global routing graphs can still be large (they often have more than 10^7 vertices). Nevertheless, global routing can be solved quite well in theory and practice. The best approach with a theoretical guarantee is based on first approximately solving a fractional relaxation (called min-max resource sharing, a generalization of multi-commodity flows), then applying randomized rounding to obtain an integral solution, and finally correcting local violations (induced by rounding).

Global routing is also done at earlier stages of the design flow, e.g. during placement, in order to estimate routability and exhibit areas with possible routing congestion.

4 Timing Optimization

A chip performs its computations in cycles. In each cycle electrical signals start from registers or chip inputs, traverse some circuits and nets, and finally enter registers or chip outputs.

Timing optimization has to ensure that all signals arrive within a given cycle time. Under this constraint, the power consumption is to be minimized. However, achieving the cycle time is al-

This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807

ready a difficult problem on its own.

4.1 Logic Synthesis

The structure of a boolean circuit has a big impact on the performance and power consumption of a chip. On the one hand the depth, i.e. the maximum number of logic circuits on a combinatorial path should be small to meet the cycle time. On the other hand the total number of circuits to realize a function should not be too big.

Almost all boolean functions have a minimum representation size which is exponential in the number of input variables. Hence, functions that are realized in hardware are quite special.

Some very special functions, such as adders, certain symmetric functions, or paths consisting alternately of AND- and OR-circuits, can be implemented optimally or near-optimally by divide-and-conquer or dynamic programming algorithms. But general logic synthesis is done by (mostly local) heuristics today.

4.2 Repeater Trees

Another central task is to distribute a signal from a source to a set of sinks. As the delay along a wire grows almost quadratically with its length, repeaters, i.e. circuits implementing the identity function or inversion, have to be inserted to strengthen the signal and linearize the growth.

For a given Steiner tree, repeaters can be inserted arbitrarily close to optimally in polynomial time using dynamic programming.

A more difficult problem asks for the structure of the Steiner tree (into which repeaters can be inserted). A minimum length Steiner tree can have very long source-sink paths. In addition, every bifurcation from a path adds capacitance and delay. Thus, trees should not only be short, but also consist of short paths with few bifurcations.

Combining approximation algorithms for minimum Steiner trees with Huffman coding, bicriteria algorithms can be derived, trading off total length and path delays.

4.3 Circuit Sizing

In *circuit sizing*, the channel widths of the underlying transistors are optimized. A wider channel charges the capacitance of the output net faster, but increases the input capacitances and, thus, the delays of the predecessors. Assuming continuously scalable circuits and a simplified delay model, the problem of finding optimum sizes for all circuits can be transformed into a geometric program. This can be solved by interior point methods or by the subgradient method and Lagrangian relaxation.

However, rounding such a continuous solution to discrete circuit sizes can corrupt the result. Theoretical models for discrete timing optimization, such as the *discrete time-cost tradeoff problem*, are not well-understood yet. Thus, local search is used extensively for post-optimization.

4.4 Clocktree Construction

One of the few problems that can be solved efficiently in theory and in practice is *clock skew scheduling*. Each register triggers its stored bit once per cycle. The times at which the signals are released can be optimized such that the cycle time is minimized. To this end a register graph is constructed. Each register is represented by a vertex. There is an arc if there is a signal path between the corresponding registers. An arc is weighted by the maximum delay of a path between the two registers. Now the minimum possible cycle time is given by the maximum mean arc weight of a cycle in the graph. This reduces to the well-solved *minimum mean cycle problem*.

The challenging problem is then to distribute a clock signal such that the optimal trigger times are met. Here *facility location* algorithms for bottom-up tree construction are combined with dynamic programming for repeater insertion.

Further Reading

1. International Technology Roadmap for Semiconductors (ITRS). SEMATECH, Austin 2013; `http://www.itrs.net` (annually updated).
2. Held, S., Korte, B., Rautenbach, D., Vygen, J.: Combinatorial optimization in VLSI design. In: "Combinatorial Optimization: Methods and Applications" (V. Chvátal, ed.), IOS Press, Amsterdam 2011, pp. 33–96.
3. Alpert, C.J., Mehta, D.P., Sapatnekar, S.S. (eds.): Handbook of Algorithms for Physical Design Automation. Taylor and Francis, Boca Raton 2009.

This paper appeared in: Princeton Companion to Applied Mathematics, (N.J.Higham, ed.), Princeton University Press 2015, 804-807