GRAPE: Genetic Routing And Placement Engine

Ronny Krashinsky
ronny@mit.edu
MIT Laboratory for Computer Science, Cambridge, MA 02139
Embodied Intelligence (6.836) Term Project

5-10-2000

Abstract

Traditional VLSI systems automatically place and route circuits using two independent steps; first an optimal placement is determined by using wire length estimates, then an optimal routing is attempted for the given placement. In actuality, placement and routing can have complicated interactions, and separating them can lead to conservative and un-optimal solutions. GRAPE, a Genetic Routing And Placement Engine, leverages the power of genetic algorithms to simultaneously optimize the placement and routing of a circuit. Using GRAPE, a population of individual placement and routing solutions evolve to maximize connectedness and minimize area.

1 Introduction

The design of modern VLSI (Very Large Scale Integration) circuits depends on an increasing amount of automation. As circuits become more complex, designers must rely on computer aided design (CAD) tools to provide a higher level of abstraction than the collection of rectangles which define an interconnected network of transistors. One task that has been delegated away from today's circuit designers is the placement and routing of components in a circuit layout.

From a bird's-eye view, a VLSI circuit is composed of various blocks[1] which are connected together with *nets*[2]. The complexity of the blocks may vary greatly, and some blocks may themselves be composed of smaller blocks. Additionally, the same type of block may be instantiated multiple times in a given circuit. At a high level, a circuit design consists of a netlist of components and the connections between them, for example Figure 1. To realize such a circuit, a designer constructs a transistor-level implementation for each component. Then, during the *layout* process, a designer places all the components on a two-dimensional grid along with the wires that connect them, for example Figure 2.

The layout process can be tedious and very difficult, especially for a certain class of irregular circuits. Such circuits are amenable to *automatic placement and routing*. In this process, a designer specifies the netlist of components along with a structural description of each component, and a software package automatically generates the circuit layout. Often, the level of abstraction is raised even higher, and a *synthesis* tool is used to translate a functional circuit description into a netlist of predefined circuit components called *standard cells*. For example, in the design of a CPU the datapath may be custom designed and manually laid out; however, the control circuitry would usually be specified as a collection of logical equations. This description is then synthesized and automatically placed and routed to achieve the final representation which goes alongside the datapath in the chip.

The standard goal of a placement and routing package is to minimize the total area used by the circuit. The constraints include the two-dimensional nature of a chip, and limited wiring resources. Traditionally, placement and routing are performed as separate steps due to the computational complexity. However, because placement and routing may have complicated interactions, the results may be conservative and un-optimal. In GRAPE, I leverage the power of genetic algorithms to simultaneously optimize placement and routing.

Below, I introduce placement and routing, and genetic algorithms in more detail. Section 2 states the details of the placement and routing problem that I consider. Section 3 describes the genetic algorithm used in GRAPE. Section 4 presents some results and optimizations, and finally section 5 concludes.

[1] *components* and *blocks* are used interchangeably in this paper
[2] *nets* and *wires* are used interchangeably in this paper

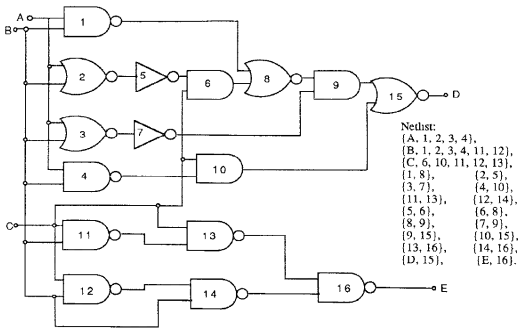

Figure 1: Example netlist. A graphical schematic of the circuit is shown along with the corresponding textual description. This figure was taken from [6].

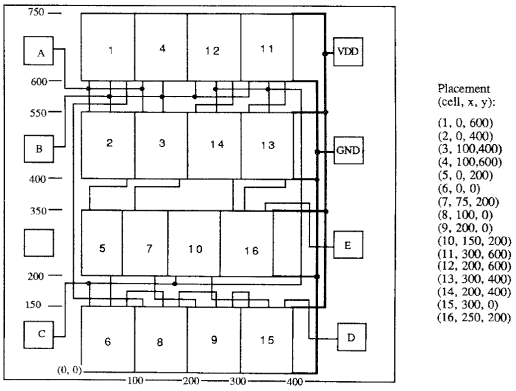

Figure 2: A placement and routing of the example circuit of Figure 1. The text shows the coordinates for each component resulting from the placement, and the figure includes the wires generated by the router. This figure was taken from [6].

1.1 Placement and Routing

The input to the placement problem is a netlist describing the connections between a set of components, for example Figure 1, along with a description of the structure of each component. The output is a set of coordinates providing a placement for each component in the x-y plane; Figure 2 shows a placement for the example netlist. The task of a placement algorithm is to minimize the total area and the estimated wire length. The method used for wire length estimation depends on the routing algorithm to be used, but a simple example is to use the Cartesian distance between connected components. There may be various additional constraints governing the placement of components depending on the requirements of the VLSI system.

Once the placement for a circuit has been determined, a routing algorithm determines the wiring for all nets between components; Figure 2 also shows the wires used to route the example circuit. The wiring constraints depend on the VLSI process parameters, but usually only horizontal and vertical wires are allowed. Typically, one layer of metal is reserved for horizontal wires, and one for vertical. In this case, perpendicular wires do not conflict with each other, unless *vias* are used to connect the wires together. Wires of the same orientation can not overlap, and there is a minimum distance requirement between parallel wires. A router must determine paths for all connections in a netlist description while minimizing wire length and area, and satisfying resource constraints. Typically, a router attempts to minimize the wire length when the output net of one component connects to several other components by using a tree structure.

Both the placement and routing problems are known to be NP-Complete [6]. Thus, it is impossible to find optimal solutions in practice, and various heuristics are used. There has been a lot of work on optimization for placement and routing, including simulated annealing algorithms, and genetic algorithms for both placement [5, 1, 4] and routing [2]. These problems are almost always solved independently, using various methods to estimate wire length during placement [6]. A simultaneous placement and global routing algorithm for FPGAs is described in [7]. To my knowledge, no simultaneous placement and routing algorithms have been proposed for VLSI circuits, and none have been proposed using genetic algorithms.

Placement algorithms are often conservative to allow sufficient resources for routing. However, there is usually no guarantee that a given placement will be routable; and even if it is, there is often no guarantee that a given routing algorithm will succeed. Thus, due

to the complicated interactions between placement and routing, they are often performed several times in an iterative manner to achieve an adequate solution. Sometimes a designer must assist the tools if automation fails.

1.2 Genetic Algorithms

Genetic algorithms [3] provide an effective means of optimization for many complex problems. Based on biological evolution, they encode a solution to a problem as a sequence of bits, or a *genome*. An initial population of genomes is established, perhaps by generating random strings of bits. This population evolves over many generations. During each generation, the individual are tested on the target task and ranked by performance. The top individuals are chosen to survive, while the rest of the population is replaced by newly constructed genomes. The key to genetic algorithms is that these new genomes are constructed by combining the genomes of the top individuals. Under the right conditions, individuals in each generation will improve by combining the beneficial properties of multiple parents, and the performance of the population at large will become more optimal.

2 Problem Statement

GRAPE uses a simplified circuit model designed to enable easy parsing and graphical display. A component is represented as a unit square with four possible ports, one at each edge. One port is designated as the output, and the other three may or may not receive inputs from other components. The output of one component may connect to any number of components, but each input port has only one connection. A circuit is placed and routed on a two-dimensional square grid.

There are no restrictions on the placement of components, except that two components can not be at the same location. The wires which connect components must be either horizontal or vertical, and perpendicular wires can overlap. There is a maximum of one horizontal and one vertical wire per unit square. A via also uses one unit square, and vias are assumed to occur in the middle of the square so that there is no interference with neighboring wires.

Figure 3 shows GRAPE placement and routing output for the example circuit of Figure 1. As the GRAPE output can be difficult to visualize, Figure 4 shows a stylized version of the same information. The simplified circuit model used by GRAPE increases the ratio of wire density to component size (1:1). A comparison between the GRAPE diagrams and Figure 2 demon-

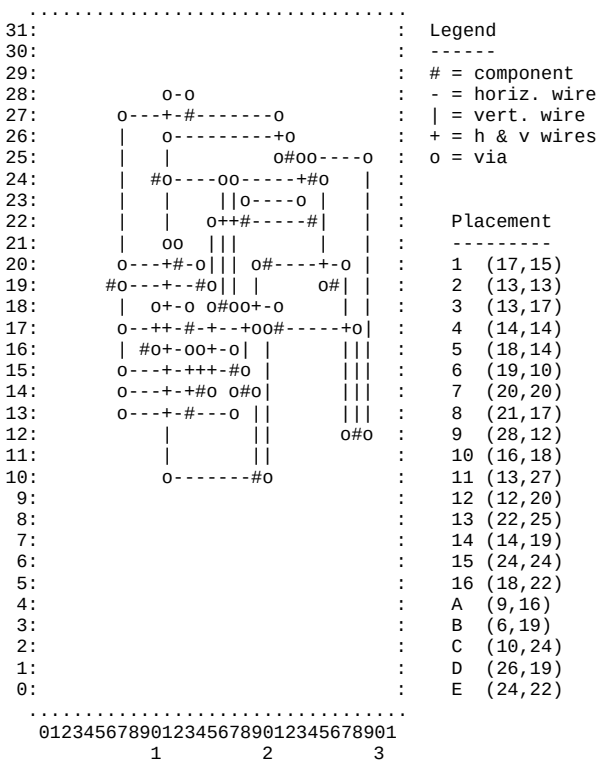

Figure 3: GRAPE placement and routing for example circuit of Figure 1. It is assumed that all components have outputs on the right side and inputs on the left and top sides.

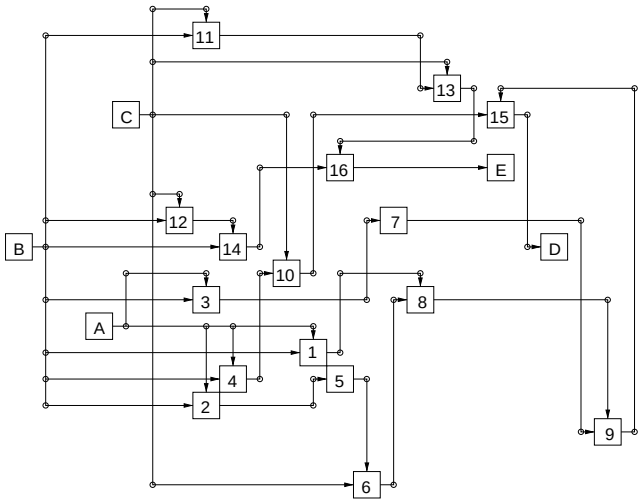

Figure 4: A stylized version of the placement and routing shown in Figure 3.

strates that this increases the wiring congestion, and thus makes placement and routing more challenging. Therefore, the circuit and routing problem solved by GRAPE can be considered a conservative approximation of that for more realistic circuits models.

3 GRAPE Genetic Algorithm

The genetic encoding used in GRAPE is shown in Figure 5. A gene consists of x and y placement coordinates, a routing algorithm specifier, and a routing ordering number. A genome is comprised of one gene for each component in a circuit. The initial population in GRAPE is constructed by generating a random genome for each individual in the population. The size of the workspace grid is configurable; in general a larger grid will enable GRAPE to achieve a complete routing faster, but optimizing for circuit area will be slower.

To determine the fitness of each individual during genetic evolution, GRAPE places and routes the input netlist based on the information in the genome. Placement is performed first by simply locating each block at the coordinates specified in its gene. If two blocks have the same coordinates, one of them is moved by increasing its x and y coordinates until a free location is found.

Once the circuit has been placed, routing is performed in the order of the components' routing numbers. For each component, all output connections are routed one at a time. The simple algorithms considered for routing a wire include using straight lines to route horizontally first, then vertically; and alternatively vertical first followed by horizontal; these are demonstrated in Figure 6. The routing algorithm specifier in the component's gene determines which routing algorithm to try first, and if that fails the other method is attempted. GRAPE also makes use of a breadth first search (BFS) algorithm; this is described in detail in the next section.

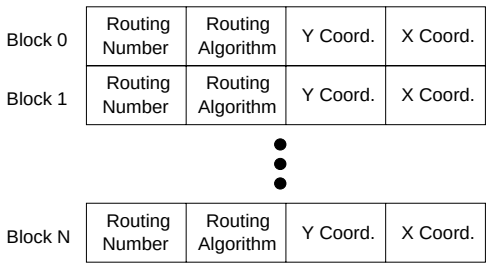

Figure 5: The genetic encoding used in GRAPE for a circuit with N blocks. The genome consists of N genes, each of which contain four parameters.

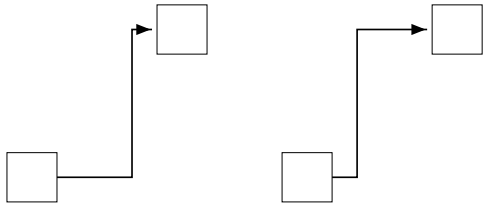

Figure 6: A comparison of horizontal-vertical and vertical-horizontal routing.

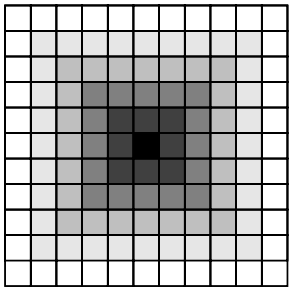

Figure 7: The scoring function used to approximate circuit density. Darker colors indicate a higher (better) score.

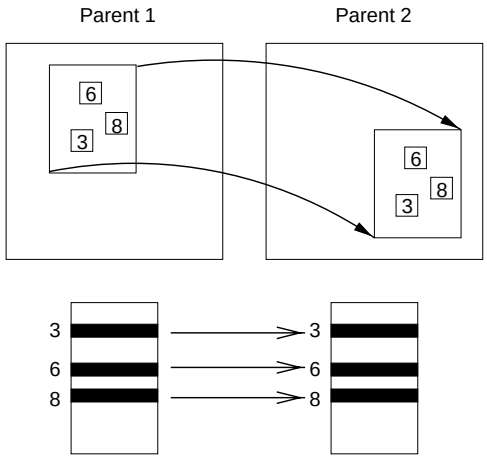

Figure 8: Genetic combination in GRAPE. A random rectangle in the placement grid of one parent is transfered to a random location in the other parent. The actual gene transfer is shown as well.

The motivation for routing all of the connections at a component's output sequentially is that the wires used can be shared. For example, in Figure 4 we can see that the output wire of component A is shared by components 1, 2, and 4. An interesting aspect of GRAPE is that it doesn't need to use special algorithms to route a net with multiple connections. When necessary, a routing naturally evolves to minimize the wiring congestion.

After placement and routing, each individual is assigned a score to determine a ranking. The most important constituent of this score is the number of connections which were successfully routed; a component is always ranked above one with a smaller routing success rate. For components with the same number of routed connections, the area and wire length used by the circuit are considered as secondary scoring metrics. For area, the actual number calculated is something more analogous to density. A certain score is given for each component, based on the function shown in Figure 7. This encourages a square placement in the center of the workspace. Wire length is calculated by summing the total number of wire segments, and this number is subtracted from an individual's total score.

After determining the top individuals in a population, offspring must be generated using genetic combination. It is important that the offspring obtain beneficial genetic material from both parents with a minimal amount of interference. Combining the genomes of two parents gene by gene in a linear manner (from top to bottom in Figure 5) proved to be insufficient for this requirement. Instead, combination in GRAPE is performed in a similar manner to [1], as shown in Figure 8. A random rectangle of the placement grid is chosen in one parent and transfered to a random location in the second parent; this involves copying the genes corresponding to any block which lies in the chosen rectangle. As described above, if two components end up being placed in the same location, one of them will be displaced. Additionally, random mutation is used to provide more variability and opportunity for optimization.

4 Results and Analysis

Figure 9 shows an example GRAPE run for a netlist of 30 components with 60 random interconnections placed and routed on a 64x64 grid. The total number of routed nets are shown along with the total wire length and density score. The wire length and density are scaled to fit on the graph. The test was run with a population of 10,000 individuals through a total of 100 generations. Each generation, 100 individuals survived and the top 10 individuals were used to produce offspring. The mutation rate was 0.2% per bit.

At the beginning of the run, the genomes evolve to create a fully connected routing of the circuit; this involves sacrificing circuit density and wire length. While attempting to route a circuit, there may be interference from components and wiring congestion. Here, the genetic combination method is critical; by using areas of the placement grid to govern inheritance, blocks of components can be moved while still maintaining relative positions for routing. Additionally, the routing ordering number and routing algorithm specifier (Figure 5) can evolve to eliminate complicated routing interactions between components. This happens largely through genetic mutation.

After the 15th generation, the circuit is fully routed and remains so for the rest of the run. During the remaining generations, the density of the circuit continues to increase while the wire length decreases. Here, the density scoring function (Figure 7) is key. A simple overall area metric would only encourage the outermost components to move inwards. The density function, in contrast, encourages all components to move towards the center of the grid. This allows the circuit to gradually compact over several generations.

With more congestion, the simple horizontal-vertical and vertical-horizontal algorithms have a difficult time routing the nets. Figure 10 shows a run where the number of components is increased to 40 with 80 random interconnections, and a decreased population of 1000. GRAPE is unable to achieve a complete routing within 100 generations.

To enable routing with more congestion, I added a breadth first search (BFS) algorithm to route nets. BFS starts at the source of a net and searches the grid for available routes, visiting each square in the grid at most once until it finds the destination component. BFS will find a shortest path if one exists, and is thus much more versatile than simple horizontal-vertical or vertical-horizontal routing. Figure 11 shows the results of using GRAPE with BFS enabled on the same circuit of 40 components and 80 random interconnections. In this run, BFS was used to route nets whenever the simpler algorithms failed; as shown in the graph this was about half the time for the best individual. A problem with using BFS is that complex routes cause more congestion; and if a circuit is fully routed, it is difficult for it to evolve into a configuration with less wiring density. Also, the running time of BFS routing can be prohibitive; the system was about 100 times slower in this example.

To make better use of BFS routing, I considered a version of GRAPE which first placed and routed

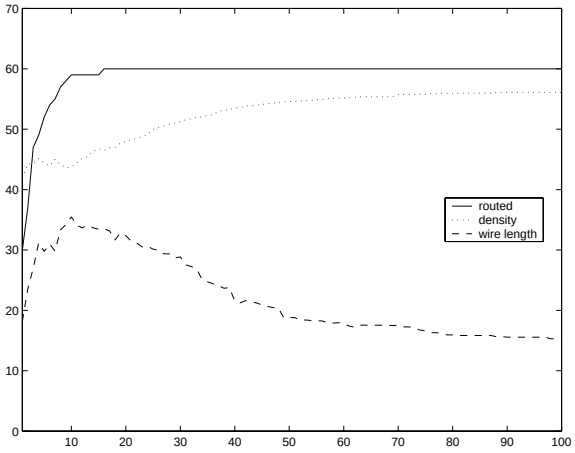

Figure 9: GRAPE placement and routing results. Components=30; Random Nets=60; Grid=64x64; Population=10,000; Generations=100; Survival rate=1%; Reproduction pool=0.1%; Mutation rate=0.2%.

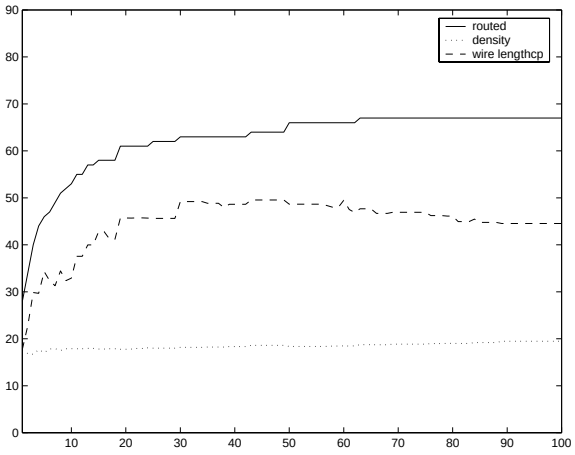

Figure 10: GRAPE placement and routing results using no BFS. Components=40; Random Nets=80; Grid=64x64; Population=1,000; Generations=100; Survival rate=10%; Reproduction pool=1%; Mutation rate=1%.

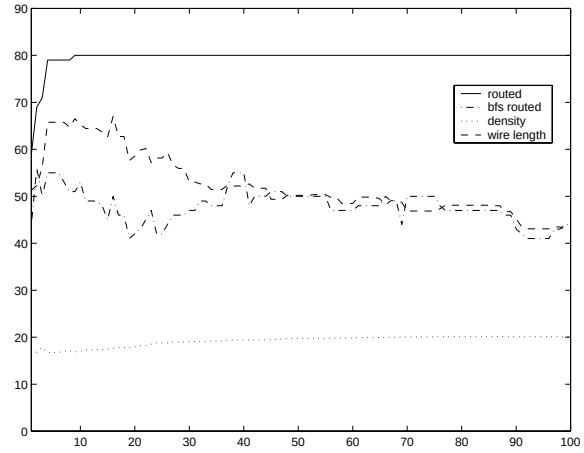

Figure 11: GRAPE placement and routing results using BFS. Components=40; Random Nets=80; Grid=64x64; Population=1,000; Generations=100; Survival rate=10%; Reproduction pool=1%; Mutation rate=1%.

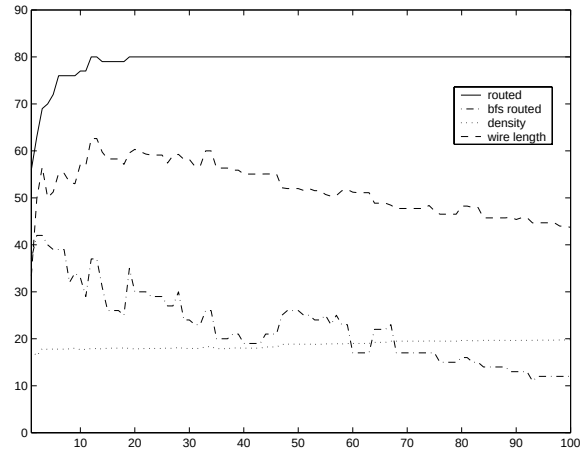

Figure 12: GRAPE placement and routing using limited BFS. Components=40; Random Nets=80; Grid=64x64; Population=1,000; Generations=100; Survival rate=10%; Reproduction pool=1%; Mutation rate=1%.

all genomes using only simple horizontal-vertical and vertical-horizontal routing paths. Then the population was ordered by performance, and only the top performing individuals were routed with BFS enabled. Figure 12 shows an example run in which BFS was enabled for 10% of the individuals. Additionally, in determining a final ranking for the population, the number of routes obtained using BFS was used as a negative component of the score. The figure shows that the population quickly achieves a complete routing through the use of BFS. However, as the population evolves, the number of nets routed using BFS decreases while the complete routing is still maintained; the total wire length and density also continue to improve. This version of GRAPE benefits from the BFS algorithm, but still can run at speeds comparable to runs which use only simple routing algorithms.

5 Conclusion

This paper presented GRAPE, a Genetic Routing And Placement Engine. Through the use of evolutionary genetic algorithms, GRAPE simultaneously optimizes both the placement and routing of a netlist of interconnected components. The genetic encoding consists of a location for each component along with routing ordering and algorithm numbers. The routing algorithms used for each net include finding strictly horizontal-vertical and vertical-horizontal paths as well as a breadth first search shortest path algorithm. The genetic system is carefully constructed to allow gradual optimization and beneficial inheritance during combination. Using GRAPE, a placement and routing for a circuit evolves to become more fully connected, have a smaller total wire length, and use less area.

References

[1] J. P. Cohoon and W. D. Paris. "Genetic Placement". In *Proceedings of the IEEE International Conference on Computer-Aided Design*, pages 422–425, 1986.

[2] H. Esbensen. "A Macro-Cell Global Router Based on Two Genetic Algorithms". In *Proc. of European Design Automation Conf. Euro-DAC*, pages 428–433, Grenoble, France, September 1994.

[3] J. H. Holland. "Adaptation in Natural and Artificial Systems". In *Univ. of Michigan Press, Ann Arbor, Mich*, 1975.

[4] R. M. Kling. "Placement by simulated evolution. Master's thesis". In *Coordinated Science Lab, College of Engr. Univ. of Illinois at Urbana-Champaign*, 1987.

[5] K. Shahookar and P. Mazumder. "A genetic approach to standard cell placement using meta-genetic parameter optimization". In *IEEE Transactions on Computer-Aided Design*, volume 9, pages 500–511, May 1990.

[6] K. Shahookar and P. Mazumder. "VLSI Cell Placement Techniques". In *ACM Computing Surveys*, volume 23, pages 143–220, June 1991.

[7] N. Togawa, M. Sato, and T. Ohtsuki. "A Performance-Oriented Simultaneous Placement and Global Routing Algorithm for Transport-Processing FPGAs". In *IEEE Trans. Fundamentals*, volume E80-A, pages 1795–1806, October 1997.