Procs. 1st IEE/IEEE Int. Conf. on GAs in Engineering Systems: Innovations and Applications, GALESIA'95, 12-14 Sep 1995, Sheffield, U.K., IEE Conf. Publication No. 414, pp. 430–435

GENETIC DESIGN OF VLSI-LAYOUTS

Volker Schnecke*, Oliver Vornberger

University of Osnabrück
Department of Math./Computer Science
D-49069 Osnabrück, Germany
{volker|oliver}@informatik.uni-osnabrueck.de
http://brahms.informatik.uni-osnabrueck.de/prakt_eng/prakt.html

Abstract — A genetic algorithm for the physical design of VLSI-chips is presented. The algorithm simultaneously optimizes the placement of the cells with the total routing. During the placement the detailed routing is done, while the global routes are optimized by the genetic algorithm. This is just opposed to the usual serial approach, where the computation of the detailed routing is the last step in the layout-design.

INTRODUCTION

The *physical design* of VLSI-chips is a very complex optimization problem, which is normally solved in various sub-steps. Because there are many interdependencies between these sub-steps it is recommendable to combine some of them. Due to the complexity only heuristic approaches like genetic algorithms can be used.

The main problem when solving 'real' applications with genetic algorithms is to find a useful genotype encoding for a single solution. Our approach uses a binary tree with additional information for each node. The main advantage here is the straightforward implementation of the genetic operators.

In the following there is a detailed description given of the classical way of solving the physical design of VLSI-chips. After this our genetic algorithm is described, which combines most of the typical sub-steps. At the end some first results are presented and some plans for extensions are given.

*The work of this author is being supported in the BMBF-Project 'HYBRID–Applications of Parallel Genetic Algorithms for Combinatorial Optimization'.

PHYSICAL VLSI-DESIGN

The design-cycle of VLSI-chips consists of different consecutive steps from high-level synthesis (functional design) to production (packaging) [9]. The *physical design* is the process of transforming a circuit description into the physical layout, which describes the position of cells and routes for the interconnections between them. Figure 1 shows a schematic representation of a layout. The main concern in the physical design of VLSI-chips is to find a layout with minimal area, further the total wirelength has to be minimized. For some critical nets there are hard limitations for the maximal wirelength.

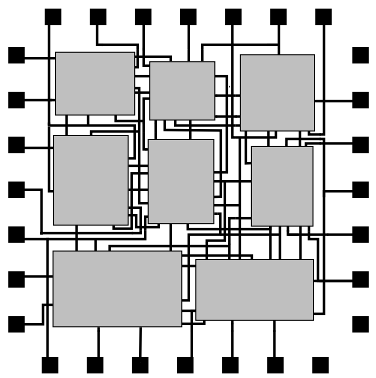

Figure 1: A layout of a chip

Due to its complexity, the physical design is normally broken in various sub-steps:

1. First the circuit has to be *partitioned* to generate some (up to 50) macro cells.
2. In the *floorplanning phase* the cells have to be placed on the layout surface.
3. After placement the *global routing* has to be done. In this step the 'loose'

routes for the interconnections between the single modules (macro cells) are determined.

4. In the *detailed routing* the exact routes for the interconnection wires in the channels between the macro cells have to be computed.

5. The last step in the physical design is the *compaction* of the layout, where it is compressed in all dimensions so that the total area is reduced.

This classical approach of the physical design is strongly serial with many interdependencies between the sub-steps. For example during floorplanning and global routing there must be enough routing space reserved to complete the exact wiring in the detailed routing phase. Otherwise the placement has to be corrected and the global routing has to be computed again.

In the following there is a closer look at steps 2. to 4., because floorplanning and routing are solved by the application described in this paper.

Floorplanning

In the floorplanning phase, the macro cells have to be positioned on the layout surface in such a manner that no blocks overlap and that there is enough space left to complete the interconnections. The input for the floorplanning is a set of modules, a list of terminals (pins for interconnections) for each module and a netlist, which describes the terminals which have to be connected. At this stage, good estimates for the area of the single macro cells are available, but their exact dimensions can still vary in a wide range. Consider for example a register file module consisting of 64 registers. It can be organized as a 1×64, 2×32, 4×16 or 8×8 array which yields four implementations with different aspect ratios. These alternatives are described by *shape-functions* [7]. A shape-function is a list of feasible height-/width-combinations for the layout of a single macro cell (cf. fig. 2). The result of the floorplanning phase is the sized floorplan, which describes the position of the cells in the layout and the chosen implementations for the flexible cells.

There exist many different approaches to the floorplanning problem. Wimer *et al.* [10]

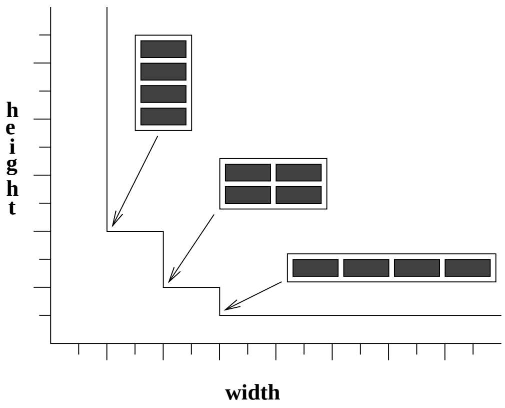

Figure 2: The shape-function for a macro cell with three different implementations

describe a branch and bound approach for the floorplan sizing problem, i. e. finding an optimal combination of all possible layout-alternatives for all modules after placement. While their algorithm is able to find the best solution for this problem, it is very time consuming, especially for real problem instances. Cohoon *et al.* [2] implemented a genetic algorithm for the whole floorplanning problem. Their algorithm makes use of estimates for the required routing space to ensure completion of the interconnections. Another more often used heuristic solution method for placement is Simulated Annealing [8, 11].

Routing

The aim of the routing phase is to find the geometrical layouts for all nets. In the floorplanning phase space on the layout surface has been provided to complete the interconnections. This space can be described as a collection of single *routing regions*. Each region has a fixed capacity, i. e. a maximum number of wires which can be routed through this region, and a number of terminals, i. e. pins on the borders of the adjacent cells.

Due to the complexity, the routing is done in two sub-phases. In the *global routing*, the 'loose' routes for the nets are determined. For the computation of the global routing, the routing space is represented as a graph, the edges of this graph represent the routing regions and are weighted with the corresponding capacities (cf. fig. 3). The global routing is described by a list of routing regions for each net of the circuit, with none of the capacities of any routing region being exceeded.

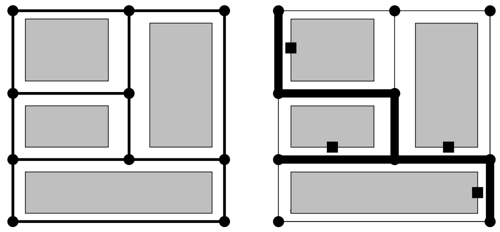

Figure 3: A routing graph (left) and a global route for a four terminal net (right)

After global routing is done, for each routing region the number of nets routed through it is known. In the detailed routing phase the exact routes for the wires have to be determined (figure 4). This is done incrementally, i. e. one channel is routed at a time in a predefined order.

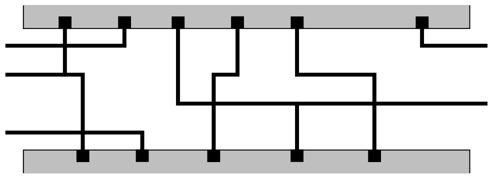

Figure 4: The detailed routing inside a channel

The global routing can be solved by graph based techniques, Integer Programming or hierarchical approaches [5]. For the detailed routing there exist solutions based on Greedy Methods, graph algorithms or hierarchical approaches [9]. The complexity and routability of a layout depends on the number of layers which can be used for the completion of the interconnections. Usually in macro cell layout there are two layers and the routing is done in the manhattan-model, i. e. one layer is for the vertical, the other for the horizontal wires and the nets change the layer when changing their direction.

THE GENETIC ALGORITHM

Our genetic algorithm combines the floorplanning with the routing phases. Because placement and even detailed routing are optimized in a single step, there is no longer the need for compaction. The main difference to the classical approach is that when building an individual, detailed routing is done during placement of the modules. The global routes and the general placement are optimized by the genetic algorithm.

The genotype

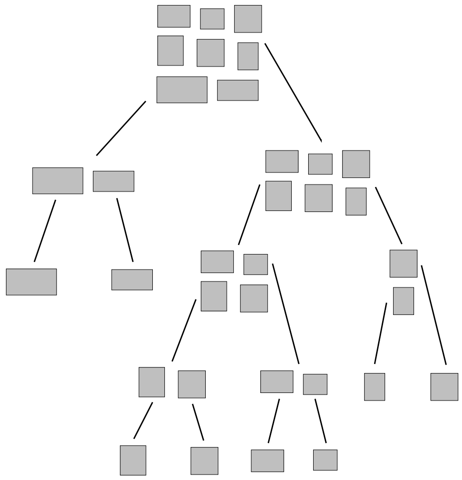

Figure 5: The slicing tree for the layout shown in figure 1

The *genotype* is encoded by a binary slicing tree with additional sizing and routing information for each node. The leaves of this tree represent the macro cells *(blocks)* and the inner nodes represent partial layouts *(meta-blocks)*, as shown in figure 5. The tree is constructed in a bottom-up fashion. When building a meta-block out of two blocks (or meta-blocks) the arrangement and the rotation of the two blocks are fixed. The exact routing in the channel between these two blocks is done (cf. fig. 6), i. e. nets between the two blocks are connected and routed. Nets which connect other terminals than those on the borders of the two blocks are passed on as terminals to the borders of the resulting meta-block.

If the blocks have different layout-alternatives, all necessary combinations of alternatives are stored in the shape-function of the resulting meta-block. The number of combinations does not grow exponentially in the number of blocks contained in a single meta-block, because some of the combi-

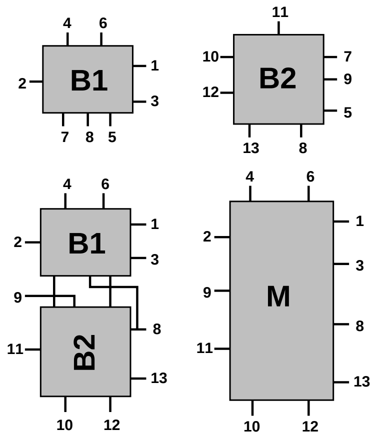

Figure 6: Combining two blocks to a meta-block

nations are redundant (cf. Fig. 7). Storing all alternatives for the meta-blocks is useful because one can not decide in the lower level of the tree, which implementation of a meta-block would be the best to minimize the overall area of the layout.

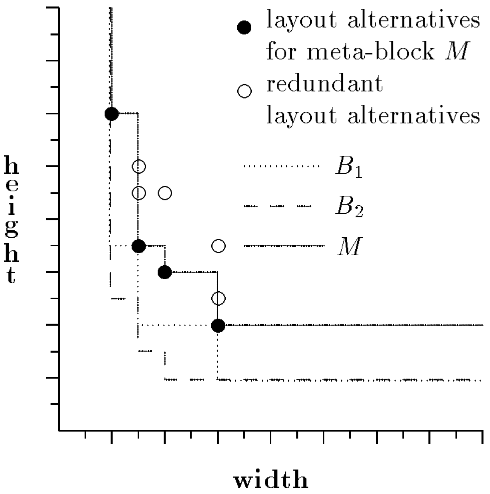

Figure 7: The construction of the shape-function for a meta-block M for block B_1 being positioned upon B_2, or vice versa

Creation of the initial population

It has proved to be useful to start with a population of not randomly created individuals. Here an iterated matching approach is implemented which yields densely packed meta-blocks in the individuals. For this, all possible pairs of modules are built and valued by the wasted space inside the resulting meta-blocks. Then a set of pairs with minimal overall waste is chosen to built the elements for the next iteration of matching. This iteration results in a set of densely packed meta-blocks with four blocks inside. The matching-process is iterated until the slicing-tree for a single individual is completed.

Mutation

There are different kinds of mutations which are applied with different frequencies. They change the structure of the slicing tree by exchanging blocks or meta-blocks and moving subtrees to other positions in the tree which corresponds to moving partial layouts on the layout surface. Further the mutation-operator rotates blocks or changes the position of two blocks inside a meta-block. Encoding different implementations for each meta-block in a genotype here enhances the performance, too, because after mutation the best implementation of a moved block in its new environment can be chosen. This might differ from the best implementation at its old position.

Crossover

The crossover-operator chooses two parent-individuals of which one offspring is produced by combining two disjunct meta-blocks (i.e. subtrees) from both individuals. It has turned out to be not helpful to do some 'intelligent' crossover, i.e. choosing densely packed meta-blocks in the parent individuals, so the meta-blocks are chosen randomly in the actual implementation. It is unlikely to get a complete layout when combining the meta-blocks, so the missing blocks have to be added. When doing this, the iterated matching method is used again to take care that the added blocks built a densely packed part in the layout which is represented by the offspring.

After the execution of a genetic operator, the slicing tree is traversed to compute the changed routing information and the shape-functions for all inner nodes.

RESULTS

	$ami33\_3$	$ami49$
# cells	33	49
# impl. per cell	3	1
# nets	123	408

Table 1: The benchmark instances

The algorithm has been tested on real-life circuits chosen from a layout-benchmark suite maintained by MCNC (North Carolina's microelectronics, computing and networking center), see table 1. The instances are layout-problems of the field of full-custom macro-cell layout.

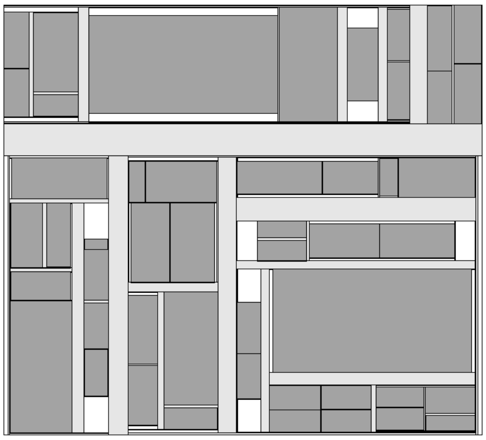

Figure 8: Layout for $ami49$ (area: $57.3\,mm^2$)

Figure 8 shows a layout for the instance $ami49$, a problem with only fixed-size cells. The dark blocks show the cells, the lighter surface represents the routing space, while the white area shows the wasted space inside the layout. The actual implementation does not do the detailed routing but adds an estimated routing space when combining two blocks. The estimation is quite bad, because actually one track is reserved for each net in the channel between two blocks. Implementing a better heuristic for the detailed routing would further reduce the channel width.

Figure 9 shows the performance of the genetic algorithm for circuit $ami49$. The best fitness averaged over 10 runs is shown for both cases up to 400 generations, the

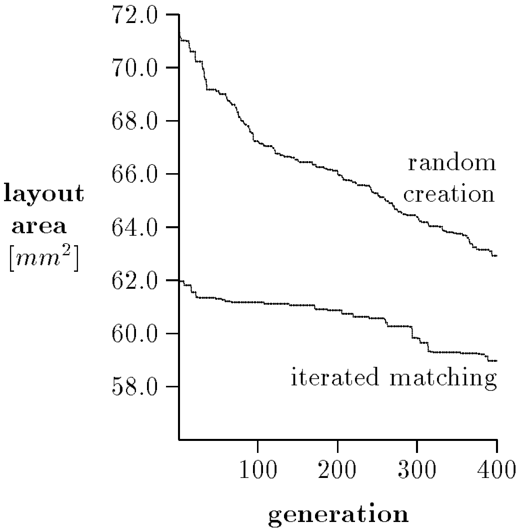

Figure 9: The performance with respect to the creation of the initial population ($ami49$)

population-size was 20 individuals. The upper curve describes the progress of the fitness (layout area) for runs of the algorithm started with populations of totally random generated individuals. The lower curve describes the performance for runs which have started with a set of very good individuals, which have been generated by application of the iterated matching. For an impression of the solution quality, which was on average $58.9mm^2$ for the 10 runs with the 'intelligent' creation of the initial population, note that the layout shown in figure 8 has an area of $57.3\,mm^2$.

For circuits with flexible cells it is helpful to store all non-redundant layout-alternatives for the meta-blocks in shape-functions to enhance the quality of the initial population. This also increases the performance of the mutation-operator, because after the movement of a meta-block its best implementation on the new position can be chosen. Figure 10 shows the benefit of storing shape-functions even for the meta-blocks for the circuit $ami33\_3$ with 33 flexible cells, each having three different implementations. It is seen that the version which stores all alternatives for the meta-blocks clearly outperformes the version of the algorithm which is described in the upper curve. Here, when pairing two flexible blocks to a meta-block, only the implementation with the minimal area is stored.

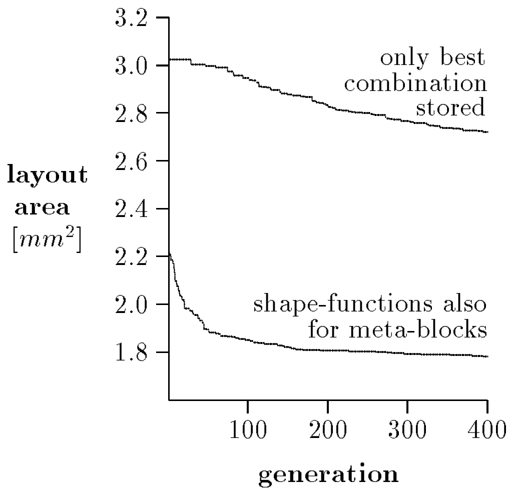

Figure 10: The benefit of shape-functions for meta-blocks, average of 10 runs (*ami*33_3)

FUTURE WORK

Future work on this project will include the computation and visualization of the exact wiring, which will replace the adding of routing-space like it is done at the moment. With this it will be possible to comprise the wirelength in the fitness, which only describes the layout area in the actual implementation. For performance enhancement the concept of *Competing Subpopulations*, introduced by Schlierkamp-Voosen and Mühlenbein [6], will be implemented. Though in this paper only results of sequential runs are presented, the algorithm is already parallelized with the use of the PVM *(Parallel Virtual Machine)* message-passing interface [4], but there will be a special effort in a more efficient parallelization.

CONCLUSIONS

An ingenious approach for the layout-design of VLSI-chips has been presented. It is contrary to the classical, serial process, where first the cells are placed and global routes for the nets are determined, before the detailed routing is done. There, global routing and floorplanning is done with having a global view on the developing layout. In the described algorithm, the detailed routing is done during the composition of partial layouts which are joined to a complete solution during the construction of the genotype, a binary slicing tree. In this approach, the global view remains to the genetic algorithm, which optimizes the general placement and the global routes on the layout surface.

References

[1] J. P. Cohoon, W. D. Paris, *Genetic Placement*, Proc. of IEEE Int. Conf. on CAD 1986, 422–425

[2] J. P. Cohoon, S. U. Hegde, W. N. Martin, D. S. Richards, *Distributed Genetic Algorithms for the Floorplan Design Problem*, IEEE Trans. on CAD, Vol. 10 (4), April 1991, 483–492

[3] H. Esbensen, *A Macro-Cell Global Router based on two Genetic Algorithms*, Proc. of European Design Automation Conf. Euro-DAC 1994, Grenoble, France, Sep. 19-23, 1994, 428–433

[4] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sunderam, *PVM: Parallel Virtual Machine*, MIT Press, 1994

[5] T. Lengauer, *Combinatorial Algorithms for Integrated Circuit Layout*, John Wiley & Sons, 1990

[6] D. Schlierkamp-Voosen, H. Mühlenbein, *Strategy Adaption by Competing Subpopulations*, 3rd Conf. on Parallel Problem Solving from Nature, Jerusalem, Israel, Oct. 9-14, 1994, Springer Lecture Notes in Computer Science 866 (1994), 199–208

[7] R. Otten, *Efficient Floorplan Optimization*, Proc. of Int. Conf. on Comp. Design 1983, 499–502

[8] C. Sechen, A. Sangiovanni-Vincentelli, *The timberwolf placement and routing package*, IEEE J. of Solid-State Circuits, Vol. SC-20 (1985), 510–522

[9] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, Kluwer Academic Publishers, 1993

[10] S. Wimer, I. Koren, I. Cederbaum, *Optimal aspect ratios of building blocks in VLSI*, 25th ACM/IEEE Design Automation Conference 1988, 66–72

[11] D. F. Wong, H. W. Leong, C. L. Liu, *Simulated Annealing for VLSI Design*, Kluwer Academic Publishers, 1988