Global routing for lookup-table based FPGAs using genetic algorithms[1]

Jorge Barreiros [1,2], Ernesto Costa [2]
[1] Departamento de Engenharia Informática e Sistemas,
Instituto Superior de Engenharia de Coimbra.
[2] Centro de Informática e Sistemas da Universidade de Coimbra.

jmsousa@isec.pt , ernesto@dei.uc.pt

Abstract. In this paper we present experiments concerning the feasibility of using genetic algorithms to efficiently build the global routing in lookup-table based FPGAs. The algorithm is divided in two steps: first, a set of viable routing alternatives is pre-computed for each net, and then the genetic algorithm selects the best routing for each one of the nets that offers the best overall global routing. Our results are comparable to other available global routers, so we conclude that genetic algorithms can be used to build competitive global routing tools.

Introduction

With the increasing growth of the complexity of electronic devices, the good performance of synthesis tools is critical for the success of design and manufacturing of non-trivial projects. Concerning the development of FPGA (*Field Programmable Gate Arrays*) systems, one of the fundamental tasks of these tools is to perform the routing of the circuit constrained to the limited resources available in the device. Although a lot of research has been made in this area [4,5,6,7,8,9,10,11,12,13], little has been done concerning the study of the feasibility of using genetic algorithms [2,3] to generate the required routing (see on [17] for a survey of genetic algorithms used for VLSI design). With this purpose, in this work we developed a global routing tool based on the application of a genetic algorithm for selection for viable routing paths on a LUT-based FPGA. Results are very close to those of other available tools, so we believe that, with further refinement, the genetic approach can became competitive with current techniques.

[1] This work was partially supported by the Portuguese Ministry of Science and High Education, under program POSI

FPGA Architecture

The FPGA architecture we used to test our algorithm is similar to the one described in [1] (see Figure 1). In this model, there are four distinct types of blocks, interconnected by several routing channels:

- **L-Blocks** – These blocks implement any logic function with N inputs. These inputs are equally distributed over all sides of the block, and a single output is provided on the right side of the block. We chose to select N=4 for all our experiments. These blocks can also be referred to as *logic cells* or *logic blocks*.
- **IO Blocks** – These blocks represent the input and output pins of the FPGA. Since the number of pins in real FPGAs is far greater than those that fit along the square formed by the L-Blocks, we consider that multiple I/O pins are contained in each I/O block. In this work, we consider that two I/O pins are available for each I/O block.
- **C Block** – These are the blocks through which the L and IO blocks are connected to routing channels. The number of alternative tracks to which a connection from a L-block or IO-block can be made, characterizes these blocks. In this work, this value was predefined to be equal to the width of the routing channel.
- **S-Block** – These blocks allow switching the tracks between different routing channels. Their *Flexibility* is defined as the number of different outputs to which a given input can be connected. We use S-Blocks with a *Flexibility* of 3 in this work (see Figure 2).

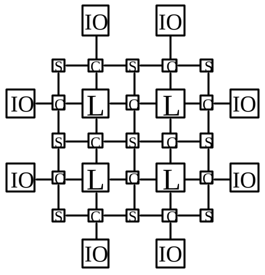

Figure 1- Architecture of the LUT-based FPGA

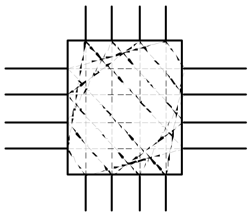

Figure 2 – S-Block configuration used in this work

Genetic Algorithms

Genetic algorithms (GA) are a group of stochastic optimization techniques [2,3]. These algorithms work wit h a set of candidate solutions to the problem (a population of individuals, using GA terminology) and seek to evolve them using concepts derived from genetics and natural selection. Each individual holds enough information (the *genes*) to describe a possible solution to the problem. They are evaluated regarding the quality of that solution (i.e. their *fitness* is computed), and a probabilistic selection method (based on the fitness of each individual) is used to find group of individuals (the *parents*) that will be used to create the next *generation* of the population. They individuals of the new generation are created by applying genetic-inspired transformations (*operators*) to the parents. Among these transformations we can find *mutations* and *crossover*. The mutation operator does random changes to the genes, while the crossover combines parts of the genes of two parents to create a single individual. After multiple iterations, the quality of the population will increase, and when a predetermined stopping condition is met, the solution for the problem will be found on the genes of the best individual of the last generation.

```
1.Randomly initialize population
2.While stopping condition is not met
     a)  Evaluate population
     b)  Select parents
     c)  Crossover
     d)  Mutation
     e)  Substitute old population
```

Figure 3 – Simple Genetic Algorithm

There are, of course, multiple variants of this simple framework.

Global Routing for FPGAs

The synthesis of circuits for FPGAs can be decomposed in several steps:

1. **Logic optimization** – The circuits are optimized, shrunk and redundant logic is eliminated.
2. **Technology Mapping –** The elements of the circuit description are assigned to specific classes of resources available in the actual FPGA. For example, logic expressions are broken into sub-expressions implemented in a single logic block.
3. **Placement –** When there are multiple components in the FPGA capable of implementing a specific aspect of the circuit description, a selection needs to be made about which one of them will be used (for example, what logic blocks will actually be used to implement the sub-expressions generated by the technology mapping?)
4. **Routing –** The communication resources available in the FPGA are used to connect adequately all the components of the circuit. Ideally, the routing should

use the smallest routing channel width and have minimum source-to-sink distance, to maximize circuit speed.

Some approaches combine some of these steps or further refine them into additional iterations. The last step is sometimes separated in global and detailed routing. The objective of the global routing step is to ensure a balanced occupation of available channels. The global router decides which routing channels will be used, without deciding about specific track usage inside those channels. Detailed routing will complete the process, by making the necessary track assignments within those channels. Frequently, it may be impossible to perform detailed routing with the same channel width found by the global routing because of restrictions on the flexibility of S-Blocks (the *routing anomaly*, see [1,4]), so the global routing channel width is actually a low bound for the width of the routing channels after detailed routing.

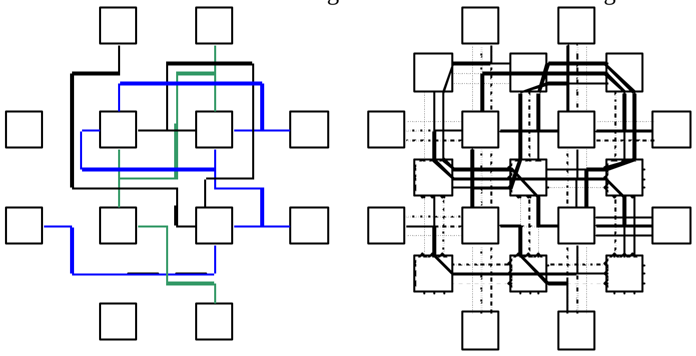

Figure 4 - Global vs. detailed routing

For further details about previous work on global and detailed routing, see [4,5,6,7,8,9,10,11,12,13]. For information regarding other steps of the synthesis process, please consult [14, 15, 16].

The Routing Algorithm

We used a two-step approach for building the global routing:

1. For each net of the circuit, generate a set of alternative routing paths.

2. Find, for all nets, the combination of alternative routing paths that offers the best overall global routing

This approach allows using different optimization techniques for each step. We developed three different heuristics for creating the alternate routing paths, and used a genetic algorithm for making the optimization described in step 2, according to the objectives of this work.

Generation of alternate routing paths

The generation of the alternate paths is made with the following algorithm:

```
For each net,

    1. Generate one routing path that connects all nodes
       in net

    2. While the desired number of alternate paths isn't
       met, build new routing path by randomly selecting
       and mutating one of the previously built paths
```

This algorithm will first generate one model net. All other alternative paths are deviations from this model net. The rationale behind this option is that it should be more efficient to build alternate solutions by making minor changes to pre-existing ones than re-computing a new net from scratch.

Computing the model routing path for each net

We developed three different algorithms for computing the first model routing. Two are graph-based search techniques and the other is based on a heuristic algorithm for computing rectilinear Steiner trees. The first graph-based technique is simply a greedy search from the source node to each sink node. This algorithm is fast but offers, as could be expected, poor quality solutions. The other graph algorithm is based on Dijkstra's algorithm [18] for finding the shortest path between two nodes. Every source/sink is connected sequentially using Dijkstra's algorithm, and the cost of previously taken segments is reduced. This algorithm offers reasonable performance, but is not as good as our heuristic for computing near optimal rectilinear Steiner trees (RST). A RST is the shortest rectilinear tree connecting a given set of points. Computing a RST instead of using graph-transversal algorithms eliminates the problem of having to route multipoint nets as groups of source/sink paths. The performance of those algorithms is very dependant on the order by which those paths are processed, and it isn't clear how to determine what the optimal order is, although some heuristic can be used (i.e. longest paths first). Our approach to computing the RST for a given set of points is based on a hill-climbing search algorithm that finds an optimal partition of those points into smaller RST, as explained bellow and illustrated in Figure 5, where a tree representation of the decomposition is presented.

The points are partitioned across horizontal and vertical axis that transverse de median point of each set, sharing one point to ensure a connected net is built. This partitioning is applied recursively until no more than a predetermined number of points (for illustration purposes, this number is 3 in Figure 5) are contained in every partition. The rectilinear tree at the nodes is built by constructing an axis along the median (vertical or horizontal) point, and connecting all other nodes to that is by transversal segments.

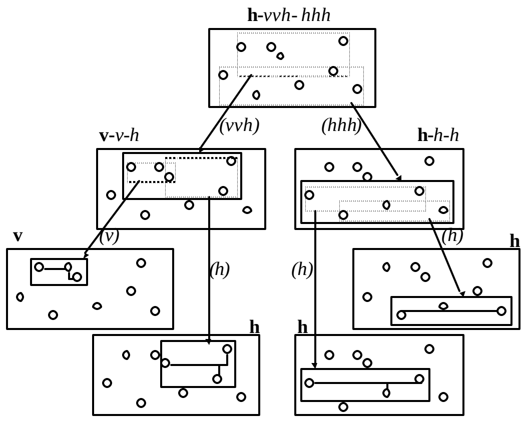

Figure 5 – Example of decomposition of point set into multiple RSTs

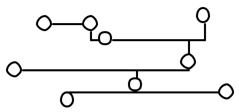

Figure 6 - (non-optimal) Rectilinear tree resulting from the decomposition illustrated in Figure 5

The actual decomposition is dependant on whether a vertical or horizontal axis is chosen on each node of the decomposition tree. This information can be represented linearly by a vector[2] where the first element represents the orientation of the axis on the root node, and the remaining information is split in half and interpreted accordingly by the left and right descendants of the root. For example, for the specific decomposition shown in Figure 5, this string is "hvvhhhh" for the root node.

The simple linear representation of the decomposition is important, because it enables us to use a simple hill-climbing optimization algorithm for finding the partition that offers the smallest length rectilinear tree. Figure 7 shows the rectilinear trees computed for a random point set with different values for maximum points per leaf. The dashed rectangles represent the points contained in a single leaf. (These trees are not optimal; they just represent a possible result of the algorithm.)

Generating mutations from model path

Deviations from a model routing path are built by applying a mutation to previously built nets. The algorithm is based on the idea of generating an intersecting rectangle over the model network and re-routing the internal connections on the borders of that rectangle. For space considerations, some details are omitted (e.g. handling nodes

[2] Although padding may be required for unbalanced decomposition trees.

internal to the rectangle, or multiple branches), but the general idea is illustrated in Figure 8.

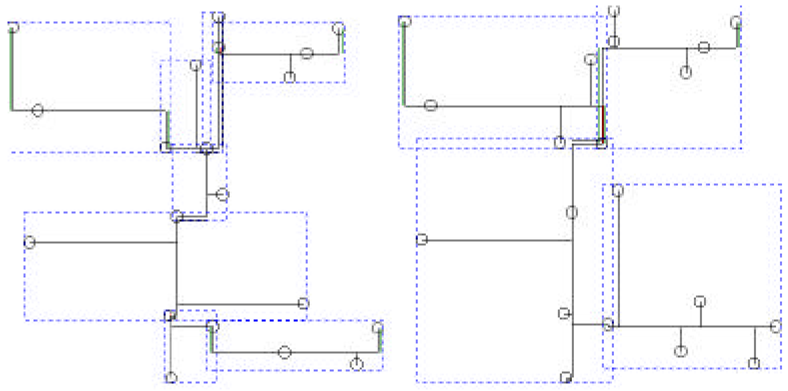

Figure 7 - Rectilinear trees for random set of points, with maximum of 4 points per leaf (left) and 8 points per leaf (right).

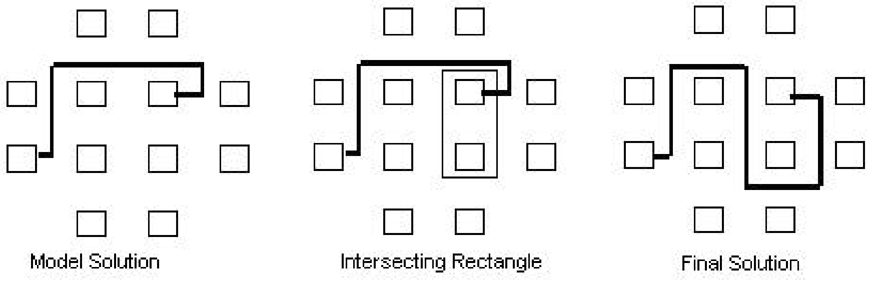

Figure 8 - Generating alternative routing paths.

Genetic Algorithm

The genetic algorithm decides among the possible alternate routes which ones will be used in the global routing. Each individual (solution) is represented only by an integer vector[3]. Each position of this vector is associated to a net of the circuit, and in that position it is held an index that identifies what is the chosen routing alternative for that net.

[3] Actually, some auxiliary data is also kept in each individual to help to speed up some computations. That detail isn't relevant to understand the main operation of the algorithm, so it is left out of the description.

After some tuning tests, the following configuration was found to offer the best results for the GA:
- Population size: 50
- Maximum number of generations: 2500
- Mutation rate: 8%
- Crossover rate:80%
- Elitism: 4 individuals
- Lamarckian learning[4]

Results

We used circuits from LGSynth93 test suite [19] to benchmark our algorithm. A set of 30 circuits of diverse complexity was chosen. The circuits were placed using VPR [10] and our tool was used to generate the global routing. For comparison purposes, VPR was also used to globally route the same circuits. A single run of both tools was used for all circuits. The test machine was a Pentium 3 processor at 866MHz with 384 MB RAMS The results are presented in Table 1.

As we can see, although VPR offers slightly better results, both in terms of CPU time and track number. However, results are reasonably close, and in some cases the AG actually computed a better solution. We believe the small difference in performance can be further reduced if other heuristics for net generation are tried and more exhaustive tuning is made.

Additionally, the AG seems to have a very large performance advantage when compared with reported results for older global routers [11], which suggests that our approach seems valid and worthy of further development.

Conclusions and future work

Our main goal was to investigate the feasibility of constructing a global router for FPGAs using genetic algorithms. We believe that our results show that this class of algorithms is worthy of consideration when designing this kind of tools. Results seem to suggest that the AG offers performance that is comparable with that of other known algorithms, although in the current implementation it is on average slightly inferior. However, this difference in performance isn't very significant, and further improvement of the algorithm may offer better results. Although some tuning tests were made, a more exhaustive set of tests might reveal some better parameterization. Also, one improvement that might significantly improve efficiency would be to generate alternative routings dynamically, only when judged necessary, instead of pre-computing a fixed-size set of alternate paths. Further work could also be done by adapting the algorithm to use architectural features of current FPGAs, like routing segments of heterogeneous size.

[4] A quick, non-exhaustive local search is conducted for each individual, and if any improvement is found it the individual is changed accordingly.

Circuit	Channel Width	Channel Width	CPU (s)	CPU (s)
	(VPR)	(AG)	(AG)	(VPR)
pdc	20	20	4299	8154
ex1010	**11**	**13**	2609	952
spla	**16**	**17**	1881	2654
s298	**9**	**10**	1247	814
seq	**13**	**14**	1158	278
alu4	11	11	807	184
misex3	12	12	661	210
ex5p	***15***	***14***	518	824
apex3	***13***	***12***	558	134
pair	**8**	**9**	476	59
C6288	6	6	89	11
i8	9	9	200	95
table5	**11**	**12**	109	33
cordic	10	10	99	68
C3540	8	9	40	27
i9	**6**	**7**	116	27
x3	6	6	273	12
vda	9	9	77	35
s1238	7	7	43	13
e64	***9***	***8***	158	45
planet	**6**	**7**	29	14
i7	**4**	**5**	12	93
mm9b	7	7	42	8
i6	5	5	13	61
alu2	7	7	33	8
too-large	**7**	**8**	8	8
C880	***8***	***7***	31	5
example2	6	6	48	3
term1	5	5	4	4
misex2	5	5	1	1
TOTAL	**269**	**277**	**15639**	**14834**

Table 1. Results for LGSynth93 test suite benchmark

[1] "On two-step routing for FPGAs", Lemieux, G. ; Brown, S. ; Vranesic, D. , International Symposium on Physical Design, Abril 1997

[2] "Genetic Algorithms in Search, optimization and machine learning", Goldber, D. Addison Wesley, 1989

[3] "An introduction to Genetic Algorithms", Mitchel, M., MIT Press, 1996

[4] "New performance-driven FPGA routing algorithms", Alexander, M. ; Robins, G; Design Automation Conference, June 1995

[5] "A detailed router for Field-Programmable gate arrays", Brown, G. ; Rose, Z. ; Vranesic, G., IEEE Transactions on Computer-Aided Design, Vol. 11, No. 5, May 1992

[6] "Plane parallel A* maze router and it's application to FPGA's"; Palczewski, M. ; Proceedings of the Design Automation Conference, 1992.

[7] "New performance-driven FPGA routing algorithms", Alexander, M. ; Robins, G; Design Automation Conference, June 1995

[8] "Performance-oriented placement and routing for Field-Programmable gate arrays", Alexander, M; Cohoon , J. ; Ganley, J. ; Robins, G., European Design Automation Conference, Sept. 1995

[9] "New performance-driven FPGA routing algorithms"; Alexander, M ; Robins , G. ; IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, Vol. 15, N. 12, Dec. 1996.

[10] "Directional bias and Non-uniformity in FPGA global routing architectures", Betz, V.; Rose, J. ; IEEE/ACM International Conference on Computer Aided Design, 1996

[11] "LocusRoute: A parallel global router for standard cells"; Rose, J. ; Proceedings of the Design Automation Conference, 1988.

[12] "A detailed routing algorithm for allocating wire segments in field-programmable gate arrays"; Lemieux, G; Brown, S; Proceedings of the ACM Physical Design Workshop, 1993.

[13] "A performance and routability driven router for FPGAs considering path delays", Lee, Y. ; Wu, A.; IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, vol 16, nº2, Feb. 1997

[14] "Optimal FPGA mapping and retiming with efficient initial state computation", Cong ; J. ; Wu, C. Proceedings of the 35th Design Automation Conference, 1998.

[15] "Technology mapping for TLU FPGA's based on decomposition of binary decision diagrams", Chang, Shih-Chieh ; Marek-Sadowska, M. ; Hwang, T., IEEE Transactions on computer aided design of integrated circuits and systems, Vol. 15, N.10, 1996

[16] "Combining technology mapping and placement for delay-minimization in FPGA designs", Chen, C.-S. ; Tsay, Y.-W. ; Hwang, T.; Wu, A. ; Lin, Y.-L.; IEEE

[17] "Genetic Algorithms for VLSI design, layout & test automation", Mazumder, P., Rudnik, E., 1999, Prentice Hall, ISBN 0-13-011566-5

[18] "A note on two problems in connection with graphs", Dijkstra, E. W.; Numerische Mathematik, vol. 1, 1959

[19] CAD Benchmarking Laboratory, North Carolina State University, LGSynth93 suite, http://www.cbl.ncsu.edu/www/