

# Gambit: A Tool for the Simultaneous Placement and Detailed Routing of Gate-Arrays<sup>\*</sup>

John Karro<sup>1</sup> and James Cohoon<sup>2</sup>

<sup>1</sup> Computer Science Program, Oberlin College, Oberlin, OH 44017  
john.karro@oberlin.edu

<sup>2</sup> Department of Computer Science, University of Virginia, Charlottesville, VA 22903  
cohoon@virginia.edu

**Abstract.** In this paper we present a new method of integrating the placement and routing stages in the physical design of channel-based architectures, and present the first implementation of this method: Gambit. Based on a graph coloring representation of the routing problem, we are able to produce circuit placements and detailed routes simultaneously, allowing routing constraints to influence decisions made in creating the placement. Gambit produces circuit mappings for both standard and three-dimensional FPGA architectures, and serves primarily as a proof-of-concept: the proposed algorithm will simultaneously perform placement and detailed routing for channel-based architectures. While the quality of Gambit mappings are not yet competitive with state-of-the-art tools in the literature, experimental results indicate that it does have the potential to become so.

## 1 Introduction

Given an abstract description of a circuit, it is no easy task to physically implement that description in hardware. The designer must map each of the components to a location on the chip, and then must run wires between component to provide the proper connections. All of this must be done with an eye towards keeping connections as small as possible, and towards minimizing the amount of chip area required for the design. The problems involved in finding an optimal solution are NP-complete, and researchers are continually developing new heuristics to produce high-quality solutions in a reasonable amount of time.

The task of physical design is frequently divided into several stages. For the popular FPGA architecture these include *technology mapping*, *placement*, *global routing* and *detailed routing*. Traditionally these have been performed in sequence, with the output of each becoming the input of the next. While such a division reduces the complexity of the problem, it has been argued that solving

---

<sup>\*</sup> Research supported by the Virginia Aerospace Consortium Graduate Fellowship Program, the National Science Foundation under grants CDA 9634333, CDR 9224789, MIP 9107717, DUE 9554715, and DUE 9653413, and by the Department of Computer Science at the University of Virginia.

the involved problems simultaneously would lead to superior results [7]. Thus there have been a number of proposals concerning the integration of stages [2, 7, 10, 11]. Many of these are aimed at integrating the placement and global routing stages; the integration of all three stages has proven considerably more difficult.

When working with channel-based architectures, the problem of detailed routing can be viewed as a graph-coloring problem [13]. Consider the Sharp methodology developed by Bapat and Cohoon [4], a technique for the integration of placement and global routing. We can integrate this graph-color representation into the methodology, allowing it to influence placement and global routing decisions. By heuristically maintaining a minimum coloring on our “routing” graph, and by making placement decisions that allow us to minimize the chromatic number of the developing graph, we can complete the Sharp algorithm and have a placement and a global routing that is guaranteed to be *fully* routable.

Gambit is our implementation of the proposed method. Developed as an extension of the Spiffy algorithm [7], a tool implementing Sharp partitioning for standard and three-dimensional FPGA architectures, Gambit produces in one step full circuit mappings that are heuristically optimized in terms of channel-width and net-length. While the results are not yet competitive against state-of-the-art tools such as Spiffy [7] or VPR [5], it is clear from the results that Gambit is on its way to becoming a state-of-the-art tool.

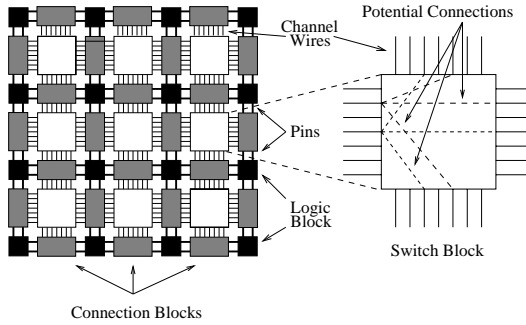
In the following paper we will discuss Gambit and the algorithm on which it is based. In Section 2 we present our problem statement and definitions, and in Section 3 we discuss details of Gambit. In Section 4 we discuss our experimental results, and in Section 5 we discuss our conclusions and the work we are currently pursuing with regard to Gambit.

It is also worth noting that while this paper concentrates on standard FPGAs, and algorithm generalizes, and has been implemented for, the newly proposed 3D-FPGAs [7, 9]. However, the techniques for doing so were straight forward, and not worth addressing here.

## 2 Problem Statement and Definitions

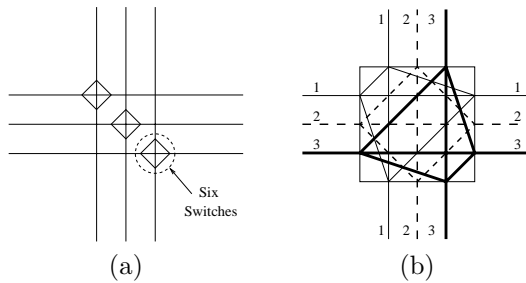
In Figure 1 we see the diagram of a Xilinx XC4000 FPGA architecture. In considering this diagram, we are concerned with five components. *Logic blocks* provide the functionality of the chip. *Channel wires* conduct signals. *Pins* carry signals from logic blocks to wires. *Connection-blocks* connect pins to wires. And *switch-blocks* connect wires. We use the term *routing blocks* to generically classify switch-blocks and connection blocks, the blocks used for signal routing. We use the term *channel* to denote the area between routing blocks through which a set of channels wires run. We denote the number of wires per channel as the *channel-width*, which must be uniform over all channels. Clearly, the smaller the channel-width, the smaller the FPGA.

In implementing a circuit on an FPGA, we program the functions of a circuit into the logic blocks, and we program the routing blocks to correctly provide connections. The challenge in doing this is to minimize the required size of



**Fig. 1.** A Xilinx XC4000 FPGA symmetrical architecture with a channel-width of seven.

the FPGA (by minimizing the required channel-width), and to maximize the performance of the FPGA (by minimizing the path connection lengths). This task is complicated by the restricted connection capacity of the switch-blocks. In order to simplify switch-block design, the designer is only allowed to connect certain subsets of wires touching the blocks. In Figure 2(a) we see the actual design of a switch-block, and in Figure 2(b) we see the possible connections such a design provides. Note that if wires are labeled sequentially from left to right or top to bottom in each channel, we have the option of connecting two wires if and only if they have the same label.

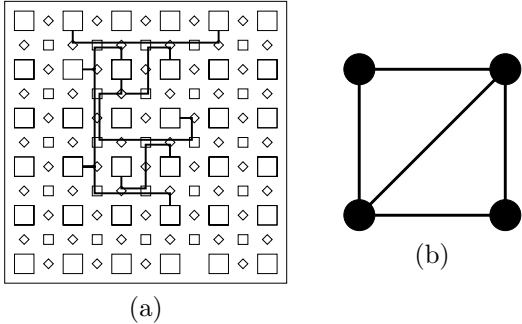


**Fig. 2.** (a) A Xilinx XC4000 switch-module and (b) its switch-module model.

While the restricted capacity of the switch-blocks complicates the design problem, it also allows us to divide the channel wires into equivalence classes. By labeling the wires in each channel as described, we know that a signal is restricted to using wires of the same label. As switch-blocks will not carry a

signal between wires of different labels, there is no way for a signal to cross these “label boundaries.” (While many routers assume that connection-blocks can be used as dog-legs, this does not seem to be the case.[5])

The existence of these equivalence classes allows the reduction of the problem of routing to a graph-coloring problem, as formulated by Wu and Marek-Sadowska [13]. Consider the global route of some circuit (that is, the assignment of signal paths to channels, but not to specific channel wires). In Figure 3(a) we see a sample global route, and in Figure 3(b) we see the corresponding *channel conflict-graph*. Such a graph has a node corresponding to each net, and two nodes share an edge if the corresponding nets share a channel in their global routes. By coloring the graph, we induce a legitimate detailed route. No two nodes that share an edge share a color, hence no two nets that share a channel are assigned to the same wire. By calculating a minimal graph coloring, we achieve an optimal detailed route given the global route.



**Fig. 3.** (a) The global routings of four nets on an FPGA. (b) The corresponding conflict-graph. Note that even though the global routings require a channel-width of 2, the conflict-graph has a chromatic number of 3 – hence the detailed routing requires a channel-width of 3.

Our objective is to simultaneously create the placement, global routing and conflict-graph coloring in such a way as to minimize the number of colors used. We begin with the block/net description of a circuit, as produced by a standard technology-mapping tool. This gives us our list of functions to be programmed into logic-blocks and our list of pins that must be connected. We then wish to simultaneously develop our placement, global route and detailed route, allowing each of the developing solutions to influence the further development of the other two solutions.

### 3 Gambit

Built on Spiffy, a tool created by Karro and Cohoon [7], Gambit uses the same divide and conquer technique. However, Gambit augments the tool with a conflict-graph structure, incrementally building the graph as a course global route is created, and then uses the graph to influence future placement and global-routing decisions. Upon completion of the algorithm, the chromatic number of the graph has been heuristically minimized, leading to a full routing with small channel-width.

We begin the Gambit algorithm by imposing a  $p \times q$  grid on the FPGA area. We partition the chip such that each logic block lies in exactly one area, with partition-lines running through switch-blocks. We then perform the following steps.

#### 3.1 Partition

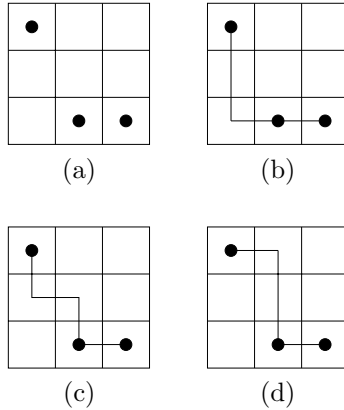
In the partition phase, we map the blocks to partitions, but not to specific logic-blocks. The goal in doing this is to minimize the number of partitions each signal must travel through in connecting the pins of a net. The stage is treated exactly the same as in the Spiffy algorithm, and simulated annealing is used to find a solution.

#### 3.2 Route Selection

Given a placement, we now need to consider each net existing in multiple partitions and choose a *thumbnail* for it: a tree that dictates the path the net will take between partitions. In Figure 4(a) we see an example of a placement of a net into partitions. In Figures 4(b)-(d) we see different *thumbnails* that can be assigned to the net; each thumbnail is a different minimum-length tree that will determine how the net signal will travel between partitions.

In the Spiffy algorithm we select a thumbnail for each net by building a vector reflecting the use of edges, and assigning thumbnails in such a way as to minimize the variance over the elements of this vector. In Gambit, we choose a thumbnail for each net, but we also assign (or reassign) a color to each node so as to conform to the constraints of the conflict graph.

We begin our algorithm by clearing the colors of all nets that will need a thumbnail, and then count the “color availability”  $\alpha(e, \kappa)$  of each thumbnail edge  $e$  and color  $\kappa$ . That is, consider thumbnail edge  $e$  and let  $S(e)$  be the set of switch-blocks laying on the corresponding partition line. If  $\mu(s)$  represents the colors of all nets currently used by switch-block  $s$  (or, more accurately, by the two sides of  $s$  relevant to the thumbnail edge), then no nets assigned to this thumbnail will be able to use any color in the set  $\cap_{s \in S(e)} \mu(s)$ . We define  $\alpha(e, \kappa)$  as the number of switch-blocks in  $S(e)$  that do not have a net of color  $\kappa$  assigned to them. Thus  $\alpha(e, \kappa)$  is the number of nets of color  $\kappa$  that we can assign to a thumbnail using edge  $e$  before  $\kappa$  becomes a member of the set  $\cap_{b \in S(e)} \mu(s)$  (the set of colors assigned to all blocks in  $S(e)$ ).



**Fig. 4.** (a) The placement of a net’s blocks with respect to the partitions. (b) One possible thumbnail (minimum-length tree) connecting the net between partitions. (c) A second possible thumbnail. (d) A third possible thumbnail.

We use a greedy algorithm in which we order the nets by number of thumbnail choices. The nets with the least number of thumbnail choices will be assigned thumbnails first, thus preventing a net with a greater number of options from using up all the choices available to nets with few options. However, where we would arbitrarily break ties in Spiffy, in Gambit we pick the longest nets. The more switch-blocks a net must use, the more restricted will be its choice of colors.

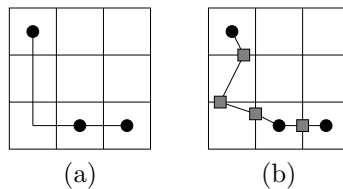
When considering a net within the order, we examine each possible thumbnail and the partition edges that thumbnail crosses. For each of these thumbnails, we determine if there is some color  $\kappa$  that can legally be assigned to the net’s conflict-graph node such that  $\alpha(e, \kappa) > 0$  for every edge  $e$  in the thumbnail. If there is no color currently used by the conflict-graph that can be assigned to the node, we eliminate the thumbnail from consideration. Only if we eliminate all thumbnails do we introduce a new color. Introducing a color increases the total number of colors used by the graph (and hence the required channel-width), but allows us to reintroduce all of the net’s thumbnails as candidates.

From this restricted set we pick our thumbnail as we did with Spiffy: in such a way so as to minimize the variance of the elements of the congestion vector for the partial solution. In addition, we assign the net  $\nu$  a color. We pick a legal color  $\kappa$  that will maximize the value  $\min_{e \in \tau(\nu)} \alpha(e, \kappa)$ , where  $\tau(\nu)$  is the set of edges used by the thumbnail of net  $\nu$ . We do this in order to avoid “using up” any color on any edge. This heuristically leaves more color options for future nets that might use of thumbnails sharing these edges. Having picked a color  $\kappa$ , we update the values of  $\alpha(e, \kappa)$  for each  $e \in \tau(\nu)$ , and repeat the process for the

next net. Upon termination of this phase, we have a thumbnail for each net and a color for each node in the conflict-graph.

### 3.3 Virtual Terminal Assignment

At this point, each net is placed with respect to the partitions, and is assigned a thumbnail that defines how its signal will travel between partitions. Now we look at the edges making up the thumbnail, and choose the exact spot on the corresponding partition line where the signal will cross. These points are known as *virtual terminals*. In Figure 5(a) we see a sample thumbnail, and in Figure 5(b) we see those points at which we could assign the net to cross the different partition lines. In an FPGA such points will take the form of switch-blocks, and define the edges of the conflict-graph. Any two nets entering the same side of a switch-block must share a channel, hence must be adjacent in the graph.



**Fig. 5.** (a) A placement and thumbnail for some net. (b) A possible assignment of virtual nodes along the thumbnail chosen for the net.

With Spiffy, the virtual terminal assignment problem is reduced to a minimum-cost perfect-matching problem and solved accordingly. The virtual terminal assignment for Gambit is very close in form to the virtual terminal assignment for Spiffy. The only additional constraints are that each net must be assigned to a switch-block that can accommodate the net's color, and that no two nets of the same color may be assigned to the same switch-block.

In the Spiffy algorithm we reduce the virtual terminal assignment to the bipartite matching problem. We do this by creating a complete bipartite graph  $P$  with a node for each net, a set of nodes for each available switch-block, and added edges between net-nodes and block-nodes. Edge weights reflect the distance between the net's terminals and the block. We ensure that no switch-block is overloaded by limiting the number of nodes in  $P$  corresponding to it.

In Gambit the limit on the number of colorings in the graph will implicitly limit switch-block overloading; there is no need to explicitly deal with the issue. Thus when creating nodes in  $P$  for each switch-block, instead of limiting the number of nodes by the ratio, we create one node for each color that can be used by the block. As the problem formulation requires that  $P$  be a complete

bipartite graph, we must add a weighted edge from each of the net-nodes to each of the block-nodes. However, for any edge that connects a net of color  $\kappa$  with a switch-block node of color other than  $\kappa$  we add an infinite penalty to the weight of the edge, thus preventing the use of that edge in the matching solution. From the manner in which we assign our net colors, we know that if there are  $k$  nets using color  $\kappa$  then there must be  $k$  switch-blocks with color  $\kappa$  available. Therefore each net can be assigned to a switch-block without incurring a penalty.

After solving the matching problem, we have the virtual nodes for each net. This has the side-effect of introducing a (heuristically) minimal number of edges into the conflict-graph. As we have been coloring the graph on the fly, it is at this point useful to re-color the graph with a more efficient heuristic, which will likely reduce the number of colors required.

Following this, we are able to apply the algorithm recursively to each of the partitions. By developing a placement and global routing within each partition, while maintaining a “global” conflict-graph, we will end up with a placement and route for the chip whose conflict-graph will immediately induce a detailed route.

### 3.4 Base Case

At some point, we must halt the recursion and explicitly solve the base case. In Spiffy, the base case is dealt with using an integer programming technique. Gambit can use a similar approach, but where the Spiffy integer program needs to determine only the exact route the nets within the base case, the Gambit integer program must additionally determine a minimum coloring of the resulting conflict-graph. Ideally we would like Gambit to fully re-color the graph at each base case, but this is computationally infeasible. Instead we must re-color only the portion of the graph affected by the base case, leading to solutions of lesser quality, but ones that can be solved in a more reasonable amount of time.

Even with this concession, the base case is Gambit’s major drawback. By adopting the base case of Spiffy to the Gambit algorithm, we have been saddled with a method that cannot handle the complexity of the total problem. In order to make Gambit into a truly competitive tool, a new approach to the base case will have to be developed.

## 4 Experimental Results

Gambit is presented as a proof-of-concept: using the conflict-graph structure, it is feasible to perform simultaneous placement and full routing. Raising the quality of Gambit’s results to a competitive level will require further research. However, our experiments indicate that such research is worthwhile.

One aspect of Gambit not yet discussed is our selection of a graph-coloring heuristic. For our first implementation, we choose two common heuristics from the literature: Brelaz’s Dsat algorithm [6], and the graph interference algorithm frequently used with compilers [1]. In Table 1 we see the channel-widths

achieved when mapping several standard benchmarks [5] to a Xilinx-4000 FPGA architecture. Results were compared against a verity of tools, including Locus-Route [12], GBP [13], TRACER [8], VPR [5], and the Alexander/Robins routing tool [3] In Table 2 we see that this version of Gambit is not competitive. (Though we note that these other tools allow dog-legging at connection blocks and thus they have an unrealistic advantage, as discussed in Section 2.)

The results do however serve as a proof-of-concept: with one tool we can generate full circuit mappings.

**Table 1.** Results of Gambit

Circuit	Dsaturn		Graph Interference	
	Average Channel-Width	Best Channel-Width	Average Channel-Width	Best Channel-Width
busc	20.47 $\pm$ 0.6	18	14.2 $\pm$ 0.58	12
dma	26.20 $\pm$ 0.57	24	17.60 $\pm$ 0.5	15
bnre	36.53 $\pm$ 0.6	33	23.63 $\pm$ 0.5	21
dfsm	32.13 $\pm$ 0.80	29	20.23 $\pm$ 0.6	18
9symml	21.43 $\pm$ 0.59	19	16.23 $\pm$ 0.6	14
term1	16.10 $\pm$ 1.1	12	11.2 $\pm$ 0.7	9
apex7	17.60 $\pm$ 0.9	14	12.97 $\pm$ 0.8	11
alu2	30.30 $\pm$ 0.78	27	21.93 $\pm$ 0.6	19
too_large	30.00 $\pm$ 0.7	27	20.03 $\pm$ 0.7	18
example2	24.60 $\pm$ 1.0	21	18.06 $\pm$ 0.9	15
vda	42.33 $\pm$ 0.7	39	28.47 $\pm$ 0.5	26
alu4	42.70 $\pm$ 0.8	38	29.16 $\pm$ 0.7	27

**Table 2.** Other Tools

Placement Tool	Altor							Spiffy
	G. Routing Tool		GBP	OCG	Tracer	VPR	A/R	
D. Routing Tool	CGA	SEGA				SEGA		Upstart
9symml	9	9	9	9	6	7	8	8
alu2	12	10	11	9	9	8	9	9
alu4	15	13	14	12	11	10	11	10
apex7	13	13	11	10	8	10	10	6
example2	18	17	13	12	10	10	11	8
term1	10	9	10	9	7	8	8	6
too_large	13	11	12	11	9	10	10	9
vda	14	14	13	11	11	12	12	10
Total	104	96	93	83	71	75	76	66

While the use of Dsaturn produces poor results, using the more sophisticated coloring algorithm within Gambit does lead to significant improvement, with

a negligible increase in run time. Clearly we have improved our results, to the point where we even best some other tools on certain benchmarks – as shown in Table 3.

**Table 3.** Gambit v. Altor

Global R.	LocusRoute		GBP	OCG	Gambit
Detailed R.	CGA	SEGA			
9symml	9	9	9	9	14
alu2	12	10	11	9	19
alu4	15	13	14	12	27
apex7	<b>13</b>	<b>13</b>	<b>11</b>	10	11
example2	<b>18</b>	<b>17</b>	13	12	15
term1	<b>10</b>	<b>9</b>	<b>10</b>	<b>9</b>	9
too_large	13	11	12	11	18
vda	14	14	13	11	26
Total	104	96	93	83	139

From these tables, we see that Gambit is highly dependent on the choice of graph-coloring techniques used. We are in the process of implementing other techniques, and have high expectations of the results that this will produce. These results will be included in the final paper.

## 5 Conclusions and Future Work

The main contribution of this paper is the introduction of Gambit, a tool significant as a proof-of-concept: conflict-graphs can be used to simultaneously perform placement and routing for channel-based architectures. By modeling the FPGA as a colorable graph, we have essentially incorporated details of the switch-block structure into the placement decisions, allowing the interaction between the formally separate stages. While results are currently not competitive, we believe that with further research, Gambit has the potential to match the results of any tool in the literature.

In working on improving Gambit’s results, we are currently pursuing two lines of research. First, we are working on a new approach to the base case, as discussed in Section 3.4. Spiffy’s approach to the base case of the recursion did not extend well to the inclusion of the conflict-graph. We intend to develop a different approach to the problem, which will lead to a better tool.

It is also evident that the choice of graph-coloring algorithm is crucial to the quality of Gambit’s results. We are currently exploring the use of more sophisticated coloring algorithms. Further, we are working on the characterization of the structure of conflict-graphs. It is our hope that we can exploit these properties shared by FPGA-induced graphs, leading to a better coloring heuristic. This in turn will lead to improved results and a faster execution time.

Our third direction of research is the generalization of Gambit to a larger class of channel-based architectures. Conceptually, the algorithm relies on the channels of the FPGA, hence it should be possible to apply the methodology to any architecture that is channel-based. Gambit exploits the “equivalence-class nature” of the channel wires, a property not necessarily shared with these other architectures. However, we are working towards a method of conflict-graph node-splitting which will allow us to compensate for that loss of structure.

Thus we present our conclusion: the use of conflict-graphs is a viable method in the integration of the placement and routing stages for channel-based architectures, and research into Gambit is worth pursuing.

## References

- [1] Aho, A.V., Sethi, R., and Jeffery, U.D. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley Publishing Company, 1986.
- [2] Alexander, M.J., Cohoon, J.P., Ganley, J.L., and Robins, G. Performance-oriented placement and routing for field-programmable gate arrays. *Proceedings of the European Design Automation Conference*, pages 80–85, 1995.
- [3] Alexander, M.J. and Robins, G. New Performance-Driven FPGA Routing Algorithms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1505–1517, December 1996.
- [4] Bapat, S. and Cohoon, J.P. A parallel VLSI circuit layout methodology. *Proceedings of Sixth International Conference on VLSI Design*, pages 236–241, 1993.
- [5] Betz, V. and Rose, J. VPR: A New Packing, Placement and Routing Tool for FPGA Research. *International Workshop on Field Programmable Logic and Applications*, pages 213–222, 1997.
- [6] D. Brelaz. New methods to color the vertices of a graph. *Communications of the ACM*, pages 251–256, 1979.
- [7] Karro, J. and Cohoon, J. A Spiffy Tool for the Simultaneous Placement and Global Routing of Three-Dimensional Field Programmable Gate Arrays. *Ninth Great Lakes Symposium on VLSI*, pages 226–227, March 1999.
- [8] Lee, Y. and Wu, A. A performance and routability driven router for FPGA considering path delays. *Proceedings of the 32 IEEE/ACM Conference on Design Automation*, pages 557–561, 1995.
- [9] Lesser, M., Meleis, W.M., Vai, M.M., and Zavracky, P.M. Rothko: A Three Dimensional FPGA Architecture, Its Fabrication, and Design Tools. *Field-Programmable Logic and Applications*, 1997.
- [10] Nag, S.K. and Rutenbar, R.A. Performance-driven simultaneous place and route for FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(5):499–518, June 1998.
- [11] Nakatake, S., Sakanushi, K., Kajitani, Y., and Kawakita, M. The channeled-BSG: a universal floorplan for simultaneous place/route with IC applications. *IEEE/ACM International Conference on Computer-Aided Design*, pages 418–25, 1998.
- [12] Rose, J. Parallel Global Routing for Standard Cells. *IEEE Transactions on Computer Aided Design*, pages 1085–1095, October 1990.
- [13] Wu, Y.L. and Marek-Sadowska, M. Orthogonal Greedy Coupling – A New Optimization Approach to 2-D FPGA Routing. *Proceedings of the ACM/IEEE Design Automation Conference*, pages 568–573, 1995.