Fast Timing-driven Partitioning-based Placement for Island Style FPGAs

Pongstorn Maidee Cristinel Ababei Kia Bazargan

Electrical and Computer Engineering Department
University of Minnesota, Minneapolis, MN 55455

{pongstor, ababei, kia}@ece.umn.edu

ABSTRACT

In this paper we propose a partitioning-based placement algorithm for FPGAs. The method incorporates simple, but effective heuristics that target delay minimization. The placement engine incorporates delay estimations obtained from previously placed and routed circuits using VPR [6]. As a result, the delay predictions during placement more accurately resemble those observed after detailed routing, which in turn leads to better delay optimization. An efficient terminal alignment heuristic for delay minimization is employed to further optimize the delay of the circuit in the routing phase. Simulation results show that the proposed technique can achieve comparable circuit delays (after routing) to those obtained with VPR while achieving a 7-fold speedup in placement runtime.

Categories and Subject Descriptors

B.7.1 [**Integrated Circuits**]: Types and Design Style—Gate arrays; B.7.2 [**Integrated Circuits**]: Design Aids—Placement and routing.

General Terms

Performance, Experimentation

Keywords

FPGAs, FPGA placement, timing-driven placement, partitioning based placement.

1. INTRODUCTION

Ideally, we would like short turnaround times in implementing complex circuits on FPGAs, achieving high frequencies. Fast synthesis and physical design algorithms targeting delay minimization are integral parts of such a design flow. Effective delay optimization on large designs is possible only by accounting for performance as early as possible in the design flow. Placement, as an early design step, should employ timing and congestion estimation and optimization techniques, to ensure small delay and routability of the circuit in the routing phase. The increased size and performance of today's Field Programmable Gate Arrays (FPGAs) allows implementation and deployment of more complex designs for dedicated applications. Large logic capacity and complex routing structures used in modern FPGAs make the development of efficient CAD tools a challenging feat.

In the last decade, there have been significant improvements in the placement and routing algorithms for ASIC and full custom designs. However, since the routing architecture in FPGAs is notably different than ASIC/full custom, physical design methods developed for these platforms cannot be directly applied to FPGAs in applications that require high performance and efficient resource utilization. There is a large degree of freedom in routing of standard cell / full custom designs, as channel widths can increase to accommodate highly congested routing regions, over-the-cell routing can be utilized to increase routing options and so on. None of these options are available in FPGAs, where the number and capacity of routing channels and internal structure of switching boxes are fixed. As a result, placement and routing of FPGAs is more challenging.

Traditionally, partitioning-based placement algorithms (e.g., [11], [13]) have been fast and hence scalable for larger design of the future. On the other hand, annealing-based placement algorithms generate high quality results at the expense of runtime (e.g., [7], [10]). It would be desirable to achieve the lower computational complexities of divide-and-conquer methods (i.e., partitioning-based / hierarchical) while obtaining the high qualities of annealing-based placement techniques.

Timing driven placement for FPGAs can be classified into two main categories: net-based and path-based approaches. Generally, path-based approaches are more accurate but slower than net-based approaches. This classification is basically the same in standard-cell placement approaches. However, differences exist in modeling the delay of the signals because of the fixed routing architecture of FPGAs.

It has been shown that the number of segments traveled by a routed net plays a more significant role in the delay of the net than the traditional geometric distance [5] [14], mainly because switches that connect routing segments dominate the delay of the segments. As a result, delay estimation based on Manhattan distance may be optimistic or pessimistic in an FPGA device with variable length segmented routing architecture [15] [16].

The differences mentioned above motivated Chang and Chang to incorporate an architecture-driven metric in their simultaneous placement and routing algorithm [14]. Better timing is obtained for circuits optimized using the new delay modeling even though

DAC 2003, June 2-6, 2003, Anaheim, CA, USA.

their results are sensitive to the order in which nets are placed and routed. Marquardt *et. al.* [7] incorporate path-based timing analysis and connection-based analysis in the simulated annealing algorithm of [6]. They obtained delay improvement at the expense of increase in wiring and run-time. In order to achieve better run-times, weighted-edge partitioning is the choice for the placement methodology proposed by Hutton [17]. A network flow based technique is used to solve the hierarchical bi-partitioning for the placement and global routing approach by Togawa [18]. There have been other previous efforts trying to develop faster placement and routing algorithms [3] [4]. Clearly, there is a tradeoff between placement runtime and the placement quality as shown by the authors in [8] [9]. Furthermore, considering routing at placement level would greatly improve the delay of the circuit after routing [19].

In this paper we propose a fast partitioning-based placement algorithm, together with a post-refinement step based on a low-temperature simulated annealing. We incorporate heuristics for delay minimization during the hierarchical placement process, which lead to circuits with the same delay compared to the state of the art VPR physical design tool for FPGAs [6]. To better understand how the VPR router behaves we first conduct some analysis on the routing of a few representative placed and routed circuits. The analysis helps us formulate better timing and congestion estimations during placement. It is important to note that the analysis is performed on a subset of circuits and the resulting data will be used in placing other circuits afterwards. Surprisingly, we noticed that VPR routing is not very sensitive to the placement algorithm, in the sense that the routing resource usage profile is similar across different placements.

Simulation results show that our proposed technique can achieve comparable circuit delays (after routing) to those obtained with VPR while achieving a 7-fold speedup in placement runtime.

The rest of this paper is organized as follows. Section 2 presents our partitioning-based placement algorithm. The terminal alignment heuristic for delay minimization is presented in Section 3. The delay model and timing criticality computations are discussed in Section 4. VPR router analysis is presented in Section 5. Simulation results are shown in Section 6. Finally, we conclude our main contribution in Section 7.

2. PPFF: PARTITIONING BASED PLACEMENT FOR FPGAS

In this section we describe our partitioning-based placement, which performs simultaneous delay and congestion minimization. Delay minimization is achieved by incorporating timing criticalities in the edge weights during partitioning, as well as terminal alignment of critical nets to reduce the number of switches used in routing the net. The terminal alignment heuristic also helps in reducing congestion. Furthermore, the partitioning engine inherently tries to reduce congestion by minimizing the cutsize at different partitioning levels.

Intuitively, the alignment of terminals of a given net has positive impact on both delay and congestion. Having more terminals along the same geometric line means a smaller number of wire segments, which translates into better delay.

The flow of our PPFF algorithm is shown in Fig. 1.

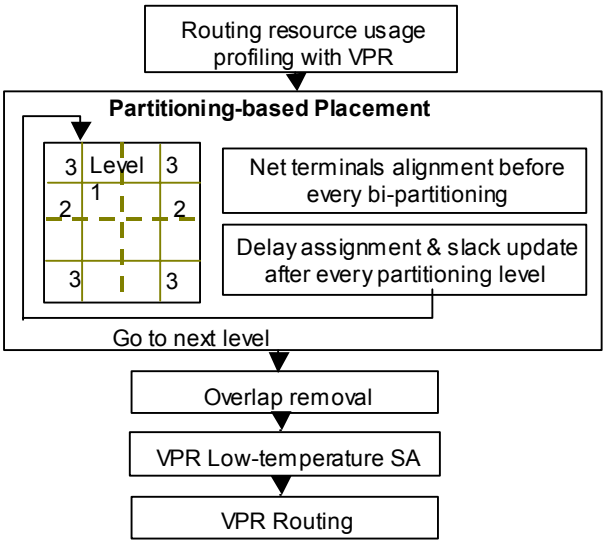

Fig. 1 Schematic diagram of proposed algorithm

Placement is done by recursive partitioning the circuit using hMetis [1]. During partitioning we maintain a tight connection between the circuit graph and placement, which represents coordinates of all cells on the FPGA fabric. This connection is key to the success of applying the net terminal alignment heuristic as well as to the accuracy of delay computations. Recursive partitioning is done until each leaf in the hierarchical partition tree contains less than a constant number of cells (e.g., six). After the recursive partitioning is completed, some leaf partitions contain more cells than they can accommodate, which in turn results in some overlaps. Overlaps are removed by using a greedy technique, which moves cells to the closest and best-aligned available empty partition.

Finally, the placement is refined by application of the VPR low temperature annealing to further minimize wire length and delay. Note that the cell swap operation in VPR is changed so that the alignments of the cells from the previous stage are not drastically reversed.

The key factors during recursive partitioning are as follows:

- Net terminal alignments (see Section 3).
- Delay and timing criticality modeling of the nets (discussed in Section 4).
- Slack assignments (covered in Section 4).

By adopting a partitioning-based approach we speed-up the placement process significantly. By using the information from the VPR router analysis we estimate timing and congestion more accurately, which results in very good circuit timing.

3. NET TERMINAL ALIGNMENT FOR DELAY MINIMIZATION

Our delay model is architecture-driven. We consider a Virtex II like multi-length segment routing architecture, which offers routing resources under the form of single-length, double-length, six-length, and long lines [2]. The traditional ASIC/full-custom measure of delay based on the geometric distance and/or channel

density is no longer accurate for segmented routing architectures of FPGAs. It is known that the number of segments used by a net is the most important factor influencing the delay [5].

Motivated by the fact that the number of segments used in routing a net is important, we implemented an efficient heuristic for delay minimization. The idea is to align the terminals of the most critical nets during the recursive partitioning. Consider nodes X and Y of a two-terminal net shown in Fig. 2. Assume that node Y is fixed (i.e., already placed at the current placement hierarchy level). Then it is more likely to use fewer routing segments to connect the two terminals if we lock node X in partition A, as opposed to partition B[1]. In this example node Y acts as an *anchor* point for net alignment. Because node Y is placed before node X, we force the partitioning process to *fix* node X in partition A. Note that if Y is not placed, then X has the freedom of being placed in either A or B, and after placement becomes the alignment anchor for Y. While simple, this heuristic proves to be very efficient and with good practical results. Experimental results demonstrate that the routing algorithm can effectively harness delay minimization opportunities created by terminal alignment during placement.

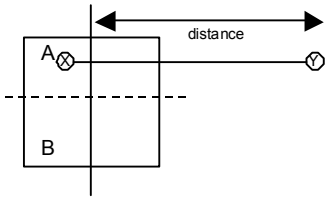

Fig. 2 Illustration of the terminal alignment of a generic two-terminal net

We adopted a recursive bi-partitioning method as opposed to quadrisection, as it makes the alignment process significantly easier to implement. However, the order in which bi-partitioning calls are issued makes the whole process look like recursive quadrisection as shown in Fig. 3.

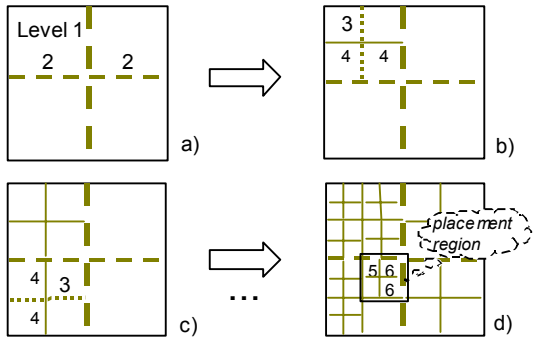

Fig. 3 Bi-partitioning seen as quadric-section

Starting with the third level of bi-partitioning (Fig. 3.b), the cut direction (horizontal or vertical) is decided based on the criticality of nets crossing the four borders of the placement region under partitioning. For example, in Fig. 4 (which is the zoom-in of the placement region shown in Fig. 3.d), the fifth bi-partitioning will be done horizontally and the two sixth level bi-partitionings will have to be done vertically. The horizontal cut direction is chosen because the largest timing criticality among nets crossing the vertical borders of the placement region (i.e., max{0.8,0.9}=0.9) is larger than that of nets crossing the horizontal borders (i.e., max{0.85,0.4}=0.85).

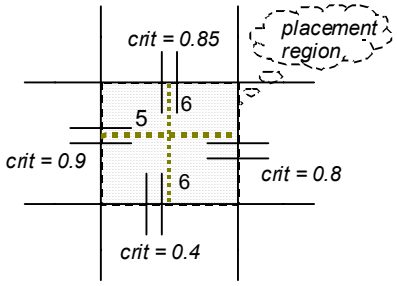

Fig. 4 Deciding the cut direction

By choosing the horizontal direction in the fifth cut we can either align or provide anchor points for the terminals of the most critical nets in this region. Fig. 5 shows an example that describes how critical nodes are aligned with nodes placed in previous placement regions. Node *x* is fixed in the bottom partition to align to the anchor that has already been placed. However, node *y* is not fixed at level 5, as the alignment direction is different from partitioning direction (it will be aligned at level 6). Nodes *z* and *t* are both free before partition 5 is performed, so the partitioning engine has the freedom in placing node *z* in either the top or the bottom partitions. However, after partitioning at level 5 in this region is done, node *z* becomes an anchor point (assuming it is a timing critical terminal), hence making terminal *t* ready to be aligned.

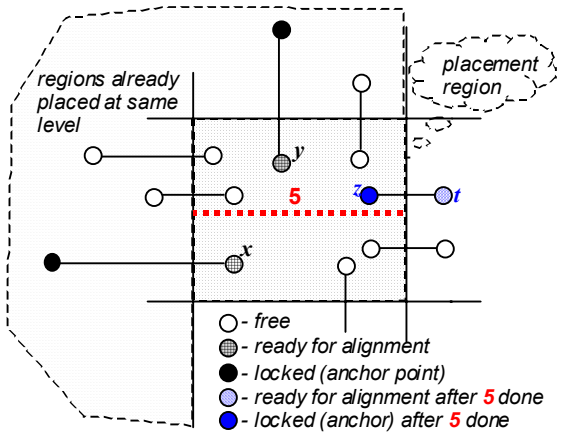

Fig. 5 Net terminal alignment step before bi-partitioning at the fifth level

To better illustrate how the partitioning is performed, Fig. 6 presents the order in which placement regions are partitioned. For every column, we start with the top placement region and continue towards bottom. The terminal alignment process continues until the last bi-partitioning at the same partitioning level is completed, following which the next level of partitioning starts. As the recursive partitioning goes on, nodes are aligned along narrower placement stripes. Terminals aligned at higher

[1] Otherwise, at least one vertical segment would be needed to change the direction of the routing from horizontal to vertical.

levels may become unaligned at lower levels if their criticality is not preserved throughout the partitioning levels.

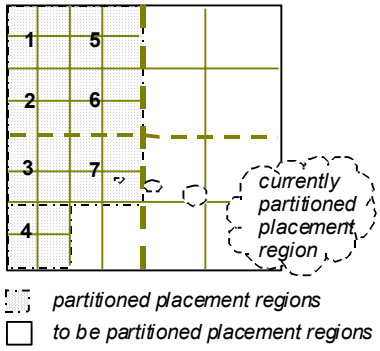

Fig. 6 Placement regions are partitioned from top-left downwards, column by column

4. DELAY MODEL AND TIMING CRITICALITY UPDATE

During recursive partitioning, edges are cut at different partitioning levels. Delay assignment to cut edges is done as follows. The minimum distance spanned by a cut net is determined by the level at which it is cut. For example, the net cut at level 5 (Fig. 7) spans a minimum distance of 3δ (where δ is the width of the smallest placement region at the current partitioning level). We cannot know the minimum distance more accurately at this stage because the *(x, y)* coordinates of the cells of this net are at the centers of the placement regions 1 and 2. At lower partitioning levels the minimum distance spanned by every net is more accurately known and hence the delay can be re-assigned with a more accurate value.

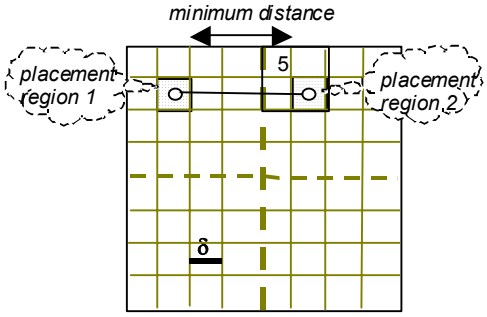

Fig. 7 A cut edge will span a minimum distance at every partitioning level

We estimate the delay of a cut net based on its timing criticality at the time of partitioning and the minimum distance it spans. Our delay model takes into account both the number of segments used for routing as well as the segment lengths. The delay value assigned to a net is the average of all delays of all nets, which span at least such a minimum distance in the final placement and routing obtained with VPR [12]. This delay value is taken from lookup-tables constructed by considering both length and criticality of all nets. That is because we are certain that in the final placement, this net will span at least that minimum distance. Details about the average delay computation will be discussed in the next section, during the VPR router analysis.

The partitioning engine incorporates a static timing analyzer (STA) used for slack computations. All edges in the circuit graph to be partitioned by hMetis are weighted. The weights represent timing criticality of the edges calculated using the timing slack values:

$$criticality_i = 1 - \frac{slack_i}{\max\limits_{alledges} slack} \qquad (1)$$

Using timing criticality as edge weights discourages the partitioning engine to cut edges with high criticalities. Therefore, critical nets will be kept short and the circuit will have a smaller delay. Nets cut at the first partitioning level are assigned delays corresponding to a single-length segment in order to ensure that these delays are the minimum delays which these nets would have after routing is finished. At the next partitioning levels, the *(x, y)* coordinates of all CLBs are more accurately known and therefore the minimum length spanned by a net is known better.

The process of delay assignment and slack/criticality update is performed at every partitioning level. Hence, timing criticalities will be more accurate and a better, tight connection between the timing-driven partitioning and placement is developed and maintained.

5. VPR ROUTER ANALYSIS

We first perform some analysis on the VPR router to better understand its behavior. This will lead to a better delay estimation and a tighter coupling between placement and routing. We start with a profiling step for the routing resource usage. We use VPR to place and route some circuits and then superimpose an imaginary grid on the FPGA fabric (Fig. 1), which represents the partitioning lines at different levels had the placement been done using a partitioning-based method as described in Section 2. The last level would consist of tiles containing a single CLB each. Nets crossing the grid lines corresponding to Level *i* are counted to get the usage of every type of routing resource at Level *i*. The characteristics of the set of combinational circuits that we used in our experiments are shown in Table I.

The key point of this step is that we noticed a common trend in the way routing resources are used by the routing tool. A typical routing resource usage is shown in

Fig. **8**. We can see that long segments are used extensively for routing nets cut at higher levels of partitioning, while double-length segments are used mostly for nets cut at lower levels. Single-length segments are used almost uniformly across all levels. The shape of these plots is preserved independently on what placement tool is used[2].

The main conclusion of the above discussion is that routing resource usage and therefore net delay is predictable. This allows us to adopt a lookup-table delay estimation technique tailored for the routing method that follows the placement. These delay lookup-tables store information about the average delay of nets with given criticality, which span a given minimum length. These tables are then used inside our partitioning-based placement

[2] We performed experiments with three different placement engines: our placement algorithm, VPR, and random placement.

algorithm for delay assignment to nets cut at different partitioning levels. It is important to note that the delay after routing is what matters in determining the performance of a circuit. Hence, if the optimizations done by the placement algorithm are in line with what the routing is inclined to do, both estimations and optimizations at placement level will eventually be more effective.

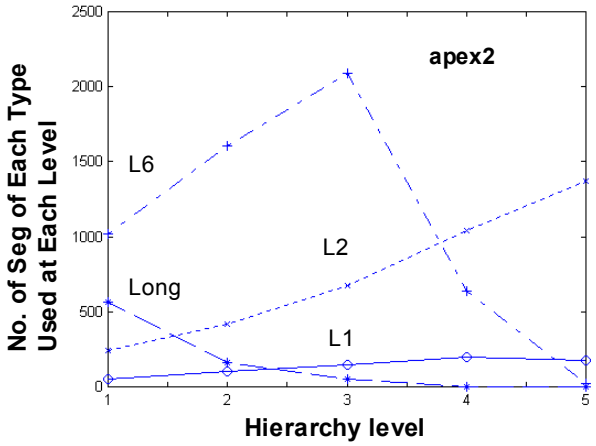

Fig. 8 Typical routing resource usage plot

To verify our intuition, we setup a simulation flow shown in Fig. 9. The goal is to study whether the information from the VPR router can be fed to the placement algorithm to achieve better timing estimates, and in turn, better final results (i.e., smaller delays after routing).

Table I Statistics of simulated circuits

Circuit	No. of CLBs	No. of I/Os	Circuit	No. of CLBs	No. of I/Os
ex5p	1064	71	seq	1750	76
apex4	1262	28	apex2	1878	42
misex3	1397	28	spla	3690	62
alu4	1522	22	pdc	4575	56
des	1591	501	ex1010	4598	20

We placed all circuits using VPR. Then we used the VPR router to route all circuits. The delay information was then used inside our partitioning-based placement algorithm (VPR delay information of each circuit is individually used for each circuit separately). Finally we study the timing of each circuit. The results are shown in Table II. It can be seen that with this simulation setup circuits placed with our placement algorithm have better delay for most of the circuits with an average of 3% improvement and as high as 9.5% for *spla*.

However, we would like to avoid repeating the VPR router analysis for each individual circuit because it requires long run-times. Instead we propose the use of average delay values of only a few representative circuits. As simulation experiments will show in the next section, this idea proves to be very effective and offers practically the same final circuit delay as obtained with VPR but at almost seven times shorter run-times.

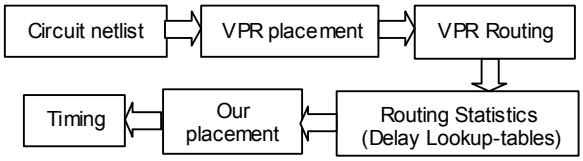

Fig. 9 Simulation setup

Table II Comparison between our results obtained with our PPFF (VPR router analysis information used for each circuit separately) and VPR

Circuit	PPFF	VPR
	Delay	Delay
ex5p	**7.69**	**8.02**
apex4	7.51	6.69
misex3	**7.25**	**7.48**
alu4	**6.72**	6.84
des	9.51	9.52
seq	8.12	8.1
apex2	8.33	8.91
spla	**12.3**	13.6
pdc	**14.4**	15.4
ex1010	**13.8**	15.1
Avg.	**0.97**	**1**

6. SIMULATION RESULTS

In this section we present simulation results obtained using our placement algorithm and compare them to those obtained with VPR, the state of the art FPGA placement tool [7]. Table III presents the simulation results. The placement algorithm uses information about the VPR router analysis as average of three different representative circuits, which are shown in shaded cells in Table I. The representative circuits were selected randomly but of all sizes.

All circuits are placed with our algorithm and with VPR and successfully routed with VPR router. It can be seen that netlists placed with our algorithm have the same average circuit delay as the ones placed with VPR, but at almost 7x faster run-times. Most circuits were routed using the best Channel Width (CW), found by VPR. Because we use the VPR profiling on a few circuits only, our algorithm can be used as a stand-alone placement tool, and hence the placement run-time for a larger set of circuits will be dramatically decreased. Note that in this situation our algorithm can also be used for multiple placement runs of the same circuit for quick solution space exploration.

Table III Comparison between our results obtained with our PPFF (VPR router analysis information used as the average of three circuits) and VPR

Circuit	PPFF			VPR		
	Delay	CW	CPU(s)	Delay	CW	CPU(s)
ex5p	7.91	23	30	8.02	**22**	177
apex4	7.5	23	35	**6.69**	23	208
misex3	8.33	20	38	7.48	**19**	227
alu4	6.84	19	41	6.84	19	245
des	**9.4**	22	60	9.52	22	382
seq	**7.66**	23	51	8.1	23	432
apex2	**7.87**	22	56	8.91	22	370
spla	13.2	30	154	13.6	30	1180
pdc	15.1	32	221	15.4	32	1599
ex1010	15.2	22	210	15.1	22	1408
Avg.	**1**	**236**	**1**	**1**	**234**	**x6.68**

7. CONCLUSIONS

We proposed a fast partitioning-based FPGA placement algorithm for delay minimization. The proposed algorithm uses delay information extracted from only a few representative placed and routed circuits. Therefore, it can be used as a fast alternative placement method for all other circuits without circuit delay degradation. Simulation experiments showed 7x shorter placement run-times, with similar circuit delays, and little area increase due to larger channel widths.

8. REFERENCES

[1] G. Karypis, R. Aggarwal, V. Kumar and S. Shekhar , "Multilevel Hypergraph Partitioning: Application in VLSI domain", *Proc. ACM/IEEE DAC*, 1997.

[2] Xilinx Inc., The Programmable Logic Data Book, 2002.

[3] P. K. Chan and M. D. F Wong , "Parallel Placement for Field-Programmable Gate Arrays", *Proc. of the Eleventh International Symposium on FPGAs*, 2003.

[4] M. G. Wrighton and A. M. DeHon , "Hardware-Assisted Simulated Annealing with Application for Fast FPGA Placement", *Proc. of the Eleventh International Symposium on FPGAs*, 2003.

[5] Y.-W Chang, K. Zhu and D. F. Wong , "Timing-Driven Routing for Symmetrical Array-Based FPGAs", *ACM Trans. on Design Automation of Electronic Systems*, vol. 5, no. 3, pp. 433-450, July 2000.

[6] V. Betz and J. Rose, "VPR: A New Packing, Placement and Routing Tool for FPGA Research", *IWFPLA* 1997.

[7] A. Marquardt, V. Betz and J. Rose, "Timing-Driven Placement for FPGAs", *FPGA* 2000.

[8] C. Mulpuri and S. Hauch , "Runtime and quality tradeoffs in FPGA Placement and routing", *Proc. of the Ninth International Symposium on FPGAs*, 2001.

[9] Y. Sankar and J. Rose , "Trading quality for compile time: ultra-fast placement for FPGAs", *Proc. of the Seventh International Symposium on FPGAs*, 1999.

[10] W. Swartz and C. Sechen , "Timing Driven Placement for Large Standard Cell Circuits", *Proc. ACM/IEEE DAC*, 1995.

[11] D. J.-H. Huang and A.B. Kahng , "Partitioning-based Standard-cell Global Placement with an Exact Objective", *Proc. ACM/IEEE ISPD*, 1997.

[12] V. Betz, J. Rose and A. Marquardt, Architecture and CAD for Deep-submicron FPGAs, Kluwer Academic Publishers, 1999.

[13] M. Wang, X. Yang and M. Sarrafzadeh , "DRAGON2000: Standard-Cell Placement Tool for Large Industry Circuits", *Proc. ACM/IEEE ICCAD*, 2000.

[14] Y.-W. Chang and Y.-T. Chang, "An Architecture-Driven Metric for Simultaneous Placement and Global Routing for FPGAs", *Proc. ACM/IEEE DAC*, 2000.

[15] M. Khellah, S. Brown and Z. Vranesic , "Minimizing Interconnection Delays in Arrays-based FPGAs", *CICC* 1994.

[16] V. Betz and J. Rose , "FPGA Routing Architecture: Segmentation and Buffering to Optimize Speed and Density", *FPGA* 1999.

[17] M. Hutton, K. Adibsamii and A. Leaver , "Timing-Driven Placement for Hierarchical Programmable Logic Devices", *FPGA* 2001.

[18] N. Togawa, M. Sato and T. Ohtsuki , "A Simultaneous Placement and Global Routing Algorithm with Path Length Constraints for Transport-Processing FPGAs", 1997.

[19] S. K. Nag and R. A. Rutenbar, "Pertbrmance-driven simultaneous placement and routing for FPGAs", *IEEE Trans. on Computer-Aided Design*, Vol. 17, No. 6, pp. 499-518, June 1998.