VPR: A New Packing, Placement and Routing Tool for FPGA Research[1]

Vaughn Betz and Jonathan Rose
Department of Electrical and Computer Engineering, University of Toronto
Toronto, ON, Canada M5S 3G4 {vaughn, jayar}@eecg.toronto.edu

Abstract

We describe the capabilities of and algorithms used in a new FPGA CAD tool, Versatile Place and Route (VPR). In terms of minimizing routing area, VPR outperforms all published FPGA place and route tools to which we can compare. Although the algorithms used are based on previously known approaches, we present several enhancements that improve run-time and quality. We present placement and routing results on a new set of large circuits to allow future benchmark comparisons of FPGA place and route tools on circuit sizes more typical of today's industrial designs.

VPR is capable of targeting a broad range of FPGA architectures, and the source code is publicly available. It and the associated netlist translation / clustering tool VPACK have already been used in a number of research projects worldwide, and should be useful in many areas of FPGA architecture research.

1 Introduction

In FPGA research, one must typically evaluate the utility of new architectural features experimentally. That is, benchmark circuits are technology mapped, placed and routed onto the FPGA architectures of interest, and measures of the architecture's quality, such as speed or area, can then readily be extracted. Accordingly, there is considerable need for *flexible* CAD tools that can target a wide variety of FPGA architectures efficiently, and hence allow fair comparisons of the architectures.

This paper describes the Versatile Place and Route (VPR) tool, which has been designed to be flexible enough to allow comparison of many different FPGA architectures. VPR can perform placement and either global routing or combined global and detailed routing. It is publicly available from http://www.eecg.toronto.edu/~jayar/software.html.

In order to make meaningful FPGA architecture comparisons, it is essential that the CAD tools used to map circuits into each architecture are of high quality. The routing phase of VPR outperforms all previously published FPGA routers for which standard benchmarks results are available, and that the combination of VPR's placer and router outperforms all published combinations of FPGA placement and routing tools.[2]

The organization of this paper is as follows. In Section 2 we describe some of the features of VPR and the range of FPGA architectures with which it may be used. In Sections 3 and 4 we describe the placement and routing algorithms. In Section 5, we compare the number of tracks required by VPR to successfully route circuits with that required by other published tools. In Section 6 we conclude and outline some future

1. This work was supported by a Walter C. Sumner Memorial Foundation Scholarship, an NSERC 1967 Scholarship, and the Information Technology Centre of Ontario.

2. Again, for those tools which have standard benchmark results to which we can compare.

enhancements which will be made to VPR.

2 Overview of VPR

Figure 1 outlines the VPR CAD flow. The inputs to VPR consist of a technology-mapped netlist and a text file describing the FPGA architecture. VPR can place the circuit, or a pre-existing placement can be read in. VPR can then perform either a global route or a combined global/detailed route of the placement. VPR's output consists of the placement and routing, as well as statistics useful in assessing the utility of an FPGA architecture, such as routed wirelength, track count, and maximum net length.

Some of the architectural parameters that can be specified in the architecture description file are:

- the number of logic block inputs and outputs,
- the side(s) of the logic block from which each input and output is accessible,
- the logical equivalence between various input and output pins (e.g. all LUT inputs are functionally equivalent),
- the number of I/O pads that fit into one row or one column of the FPGA, and
- the dimensions of the logic block array (e.g. 23 x 30 logic blocks).

In addition, if global routing is to be performed, one can also specify:

- the relative widths of horizontal and vertical channels, and
- the relative widths of the channels in different regions of the FPGA.

Finally, if combined global and detailed routing is to be performed, one also specifies:

- the switch block [1] architecture (i.e. how the routing tracks are interconnected),
- the number of tracks to which each logic block input pin connects (F_c [1]),
- the F_c value for logic block outputs, and
- the F_c value for I/O pads.

The current architecture description format does not allow segments that span more than one logic block to be included in the routing architecture, but we are presently adding this feature. Adding new routing architecture features to VPR is relatively easy, since VPR uses the architecture description to create a routing resource graph. Every routing track and every pin in the architecture becomes a node in this graph, and the graph edges represent the allowable connections. The router, graphics visualiza-

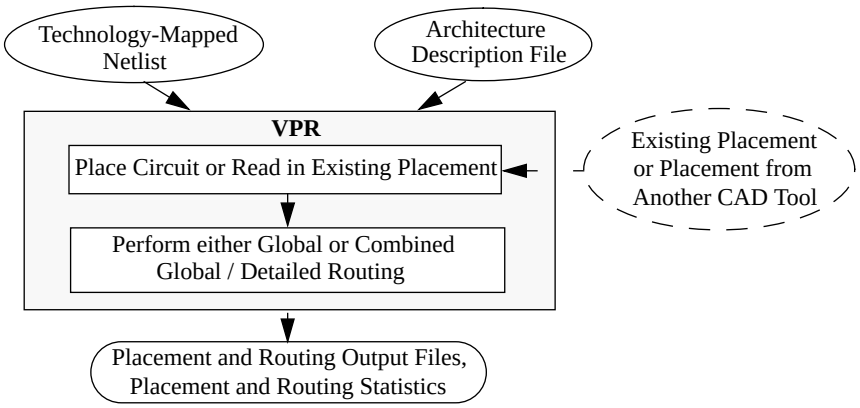

Fig. 1. CAD flow.

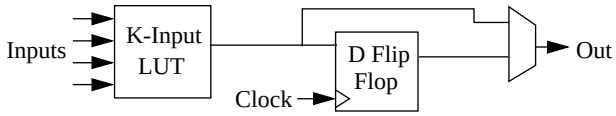

Fig. 2. Basic FPGA logic block.

tion and statistics computing routines all work only with this routing resource graph, so adding new routing architecture features only involves changing the subroutines that build this graph.

Although VPR was initially developed for island-style FPGAs [2, 3], it can also be used with row-based FPGAs [4]. VPR is not currently capable of targeting hierarchical FPGAs [5], although adding an appropriate placement cost function and the required routing resource graph building routines would allow it to target them.

Finally, VPR's built-in graphics allow interactive visualization of the placement, the final routing, the available routing resources and the possible ways of interconnecting the routing resources.

2.1 The VPACK Logic Block Packer / Netlist Translator

VPACK reads in a blif format netlist of a circuit that has been technology-mapped to LUTs and flip-flops, packs the LUTs and flip flops into the desired FPGA logic block, and outputs a netlist in VPR's netlist format. VPACK can target a logic block consisting of one LUT and one FF, as shown in Figure 2, as this is a common FPGA logic element. VPACK is also capable of targeting logic blocks that contain several LUTs and several flip flops, with or without shared LUT inputs [6]. These "cluster-based" logic blocks are similar to those employed in recent FPGAs by Altera, Xilinx and Lucent Technologies.

3 Placement Algorithm

VPR uses the simulated annealing algorithm [7] for placement. We have experimented with several different cost functions, and found that what we call a *linear congestion* cost function provides the best results in a reasonable computation time [8]. The functional form of this cost function is

$$Cost = \sum_{n=1}^{N_{nets}} q(n)\left[\frac{bb_x(n)}{C_{av,x}(n)} + \frac{bb_y(n)}{C_{av,y}(n)}\right]$$

where the summation is over all the nets in the circuit. For each net, bb_x and bb_y denote the horizontal and vertical spans of its bounding box, respectively. The q(n) factor compensates for the fact that the bounding box wire length model underestimates the wiring necessary to connect nets with more than three terminals, as suggested in [10]. Its value depends on the number of terminals of net n; q is 1 for nets with 3 or fewer terminals, and slowly increases to 2.79 for nets with 50 terminals. $C_{av,x}(n)$ and $C_{av,y}(n)$ are the average channel capacities (in tracks) in the x and y directions, respectively, over the bounding box of net n.

This cost function penalizes placements which require more routing in areas of the FPGA that have narrower channels. All the results in this paper, however, are obtained with FPGAs in which all channels have the same capacity. In this case C_{av} is a con-

stant and the linear congestion cost function reduces to a bounding box cost function.

A good annealing schedule is essential to obtain high-quality solutions in a reasonable computation time with simulated annealing. We have developed a new annealing schedule which leads to very high-quality placements, and in which the annealing parameters automatically adjust to different cost functions and circuit sizes. We compute the initial temperature in a manner similar to [11]. Let N_{blocks} be the total number of logic blocks plus the number of I/O pads in a circuit. We first create a random placement of the circuit. Next we perform N_{blocks} moves (pairwise swaps) of logic blocks or I/O pads, and compute the standard deviation of the cost of these N_{blocks} different configurations. The initial temperature is set to 20 times this standard deviation, ensuring that initially virtually any move is accepted at the start of the anneal.

As in [12], the default number of moves evaluated at each temperature is $10 \cdot (N_{blocks})^{1.33}$. This default number can be overridden on the command line, however, to allow different CPU time / placement quality tradeoffs. Reducing the number of moves per temperature by a factor of 10, for example, speeds up placement by a factor of 10 and reduces final placement quality by only about 10%.

When the temperature is so high that almost any move is accepted, we are essentially moving randomly from one placement to another and little improvement in cost is obtained. Conversely, if very few moves are being accepted (due to the temperature being low and the current placement being of fairly high quality), there is also little improvement in cost. With this motivation in mind, we propose a new temperature update schedule which increases the amount of time spent at temperatures where a significant fraction of, but not all, moves are being accepted. A new temperature is computed as $T_{new} = \alpha\ T_{old}$, where the value of α depends on the fraction of attempted moves that were accepted (R_{accept}) at T_{old}, as shown in Table 1.

Table 1. Temperature update schedule.

Fraction of moves accepted (R_{accept})	α
$R_{accept} > 0.96$	0.5
$0.8 < R_{accept} \leq 0.96$	0.9
$0.15 < R_{accept} \leq 0.8$	0.95
$R_{accept} \leq 0.15$	0.8

Finally, it was shown in [12, 13] that it is desirable to keep R_{accept} near 0.44 for as long as possible. We accomplish this by using the value of R_{accept} to control a range limiter -- only interchanges of blocks that are less than or equal to D_{limit} units apart in the x and y directions are attempted. A small value of D_{limit} increases R_{accept} by ensuring that only blocks which are close together are considered for swapping. These "local swaps" tend to result in relatively small changes in the placement cost, increasing their likelihood of acceptance. Initially, D_{limit} is set to the entire chip. Whenever the temperature is reduced, the value of D_{limit} is updated according to $D_{limit}^{new} = D_{limit}^{old} \cdot (1 - 0.44 + R_{accept}^{old})$, and then clamped to the range $1 \leq D_{limit} \leq$

maximum FPGA dimension. This results in D_{limit} being the size of the entire chip for the first part of the anneal, shrinking gradually during the middle stages of the anneal, and being 1 for the low-temperature part of the anneal.

Finally, the anneal is terminated when $T < 0.005 * Cost / N_{nets}$. The movement of a logic block will always affect at least one net. When the temperature is less than a small fraction of the average cost of a net, it is unlikely that any move that results in a cost increase will be accepted, so we terminate the anneal.

4 Routing Algorithm

VPR's router is based on the Pathfinder negotiated congestion algorithm [14, 8]. Basically, this algorithm initially routes each net by the shortest path it can find, regardless of any overuse of wiring segments or logic block pins that may result. One iteration of the router consists of sequentially ripping-up and re-routing (by the lowest cost path found) every net in the circuit. The cost of using a routing resource is a function of the current overuse of that resource and any overuse that occurred in prior routing iterations. By gradually increasing the cost of oversubscribed routing resources, the algorithm forces nets with alternative routes to avoid using oversubscribed resources, leaving only the net that most needs a given resource behind.

For the experimental results in this paper we set the maximum number of router iterations to 45; if a circuit has not successfully routed in a given number of tracks in 45 iterations it is assumed to be unroutable with channels of that width. To avoid overly circuitous routes and to save CPU time, we allow the routing of a net to go at most 3 channels outside the bounding box of the net terminals.

One important implementation detail deserves mention. Both the original Pathfinder algorithm and VPR's router use Dijkstra's algorithm (i.e. a maze router [15]) to connect each net. For a k terminal net, the maze router is invoked k-1 times to perform all the required connections. In the first invocation, the maze routing wavefront expands out from the net source until it reaches any one of the k-1 net sinks. The path from source to sink is now the first part of this net's routing. The maze routing wavefront is emptied, and a new wavefront expansion is started from the entire net routing found thus far. After k-1 invocations of the maze router all k terminals of the net will be connected.

Unfortunately, this approach requires considerable CPU time for high-fanout nets. High-fanout nets usually span most or all of the FPGA. Therefore, in the latter invocations of the maze router the partial routing used as the net source will be very large, and it will take a long time to expand the maze router wavefront out to the next sink. Fortunately there is a more efficient method. When a net sink is reached, add all the routing resource segments required to connect the sink and the current partial routing to the wavefront (i.e. the expansion list) with a cost of 0. Do not empty the current maze routing wavefront; just continue expanding normally. Since the new path added to the partial routing has a cost of zero, the maze router will expand around it at first. Since this new path is typically fairly small, it will take relatively little time to add this new wavefront, and the next sink will be reached much more quickly than if the entire wavefront expansion had been started from scratch. Figure 3 illustrates the difference graphically.

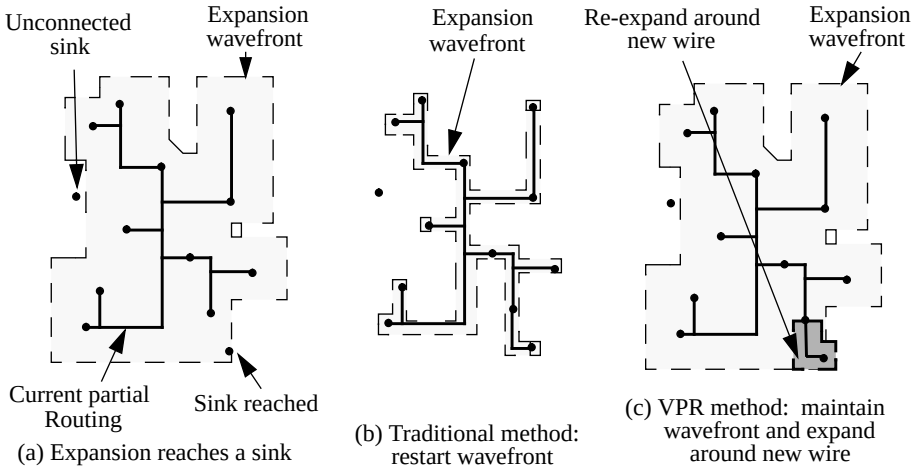

Fig. 3. When a sink is reached (a), a new wavefront can be built from scratch (b), or incrementally (c).

5 Experimental Results

The various FPGA parameters used in this section were always chosen to allow a direct comparison with previously published results. All the results in this section were obtained with a logic block consisting of a 4-input LUT plus a flip flop, as shown in Figure 2. The clock net was not routed in sequential circuits, as it is usually routed via a dedicated routing network in commercial FPGAs. Each LUT input appears on one side of the logic block, while the logic block output is accessible from both the bottom and right sides, as shown in Figure 4. Each logic block input or output can connect to any track in the adjacent channel(s) (i.e. F_c = W). Each wire segment can connect to three other wiring segments at channel intersections (i.e F_s = 3) and the switch box topology is "disjoint" -- that is, a wiring segment in track 0 connects only to other wiring segments in track 0 and so on.

5.1 Experimental Results with Input Pin Doglegs

Most previous FPGA routing results have assumed that "input pin doglegs" are possible. If the connection box between an input pin and the tracks to which it connects consists of F_c independent pass transistors controlled by F_c SRAM bits, it is possible to turn on two of these switches in order to electrically connect two tracks via the input pin. We will refer to this as an input pin dogleg. Commercial FPGAs, however,

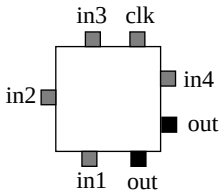

Fig. 4. Logic block pin locations.

implement the connection box from an input pin to a channel via a multiplexer, so only one track may be connected to the input pin. Using a multiplexer rather than independent pass transistors saves considerable area in the FPGA layout. As well, normally there is a buffer between a track and the connection block multiplexers to which it connects in order to improve speed; this buffer also means that input pin doglegs can not be used. Therefore, while we allow input pin doglegs in this section in order to make a fair comparison with past results, it would be best if in the future FPGA routers were tested without input pin doglegs.

In this section we compare the minimum number of tracks per channel required for a successful routing by various CAD tools on a set of 9 benchmark circuits.[1] All the results in Table 2 are obtained by routing a placement produced by Altor [16], a min-cut based placement tool. Three of the columns consist of two-step (global then detailed) routing, while the other routers perform combined global and detailed routing. VPR requires 10% fewer tracks than the second best router, and the third best router consists of VPR's global route phase plus SEGA for detailed routing.

Table 2. Tracks required to route placements generated by Altor.

Global R.	LocusRoute [17]		GBP [20]	OGC [21]	IKMB [22]	VPR	TRACER [24]	**VPR**
Detail R.	CGE [18]	SEGA [19]				SEGA [23]		
9symml	9	9	9	9	8	7	6	**6**
alu2	12	10	11	9	9	8	9	**8**
alu4	15	13	14	12	11	10	11	**9**
apex7	13	13	11	10	10	10	8	**8**
example2	18	17	13	12	11	10	10	**9**
k2	19	16	17	16	15	14	14	**12**
term1	10	9	10	9	8	8	7	**7**
too_large	13	11	12	11	10	10	9	**8**
vda	14	14	13	11	12	12	11	**10**
Total	123	112	110	99	94	89	85	**77**

Table 3 lists the number of tracks required to implement these benchmarks when new CAD tools are allowed to both place and route the circuits. The size column lists the number of logic blocks in each circuit. VPR uses 13% fewer tracks when it performs combined global and detailed routing than it does when SEGA is used to perform detailed routing on a a VPR-generated global route. FPR, which performs placement and global routing simultaneously in an attempt to improve routability, requires 87% more total tracks than VPR. Finally, allowing VPR to place the circuits instead of forcing it to use the Altor placements reduces the number of tracks VPR requires to route them by 40%, indicating that VPR's simulated annealing based placer is considerably better than the Altor min-cut placer.

5.2 Experimental Results Without Input Pin Doglegs

Table 4 compares the performance of VPR with that of the SPLACE/SROUTE tool

1. These benchmarks are available for download at http://www.eecg.toronto.edu/~lemieux/sega.

Table 3. Tracks required to place and route circuits.

Placement / Global Routing / Detailed Routing	Number of Logic Blocks in Circuit	FPR [25]	VPR / SEGA	**VPR**
9symml	70	9	6	**5**
alu2	143	10	7	**6**
alu4	242	13	8	**7**
apex7	77	9	5	**4**
example2	120	13	5	**5**
k2	358	17	10	**9**
term1	54	8	5	**5**
too_large	148	11	7	**6**
vda	208	13	9	**8**
Total	--	103	62	**55**

set, which does not allow input pin doglegs. When both tools are only allowed to route an Altor-generated placement VPR requires 13% fewer tracks than SROUTE. When the tools are allowed to both place and route the circuits, VPR requires 29% fewer tracks than the SPLACE/SROUTE combination. Both VPR and SPLACE are based on simulated annealing. We believe the VPR placer outperforms SPLACE partially because it handles high-fanout nets more efficiently, allowing more moves to be evaluated in a given time, and partially because of a more efficient annealing schedule.

Table 4. Tracks required to place and route circuits with no input doglegs.

Placement	Altor		SPLACE [26]	**VPR**
Global + Detailed Route	SROUTE [26]	**VPR**	SROUTE	
9symml	7	**6**	7	**5**
alu2	9	**8**	8	**6**
alu4	12	**10**	9	**7**
apex7	9	**9**	6	**4**
example2	11	**10**	7	**5**
k2	15	**14**	11	**9**
term1	8	**7**	5	**4**
too_large	11	**9**	8	**7**
vda	12	**10**	10	**8**
Total	94	**83**	71	**55**

5.3 Experimental Results on Large Circuits

The benchmarks used in Sections 5.1 and 5.2 range in size from 54 to 358 logic blocks, and accordingly are too small to be very representative of today's FPGAs. Therefore, in this section we present experimental results for the 20 largest MCNC benchmark circuits [27], which range in size from 1047 to 8383 logic blocks. We use Flowmap [28] to technology map each circuit to 4-LUTs and flip flops, and VPACK to

combine flip flops and LUTs into our basic logic block. The number of I/O pads that fit per row or column is set to 2, in line with current commercial FPGAs. Each circuit is placed and routed in the smallest square FPGA which can contain it. Input pin doglegs are not allowed. Note that three of the benchmarks, bigkey, des, and dsip, are pad-limited in the FPGA architecture assumed.

Table 5. Channel widths required to place and route 20 large benchmark circuits.

Circuit	# LBs	SEGA	**VPR**	Circuit	# LBs	SEGA	**VPR**	Circuit	# LBs	SEGA	**VPR**
alu4	1522	16	**10**	dsip	1370	9	**7**	s298	1931	18	**7**
apex2	1878	20	**11**	elliptic	3604	16	**10**	s38417	6406	10	**8**
apex4	1262	19	**12**	ex1010	4598	22	**10**	s38584.1	6447	12	**9**
bigkey	1707	9	**7**	ex5p	1064	16	**13**	seq	1750	18	**11**
clma	8383	≥ 24	**12**	frisc	3556	18	**11**	spla	3690	26	**13**
des	1591	11	**7**	misex3	1397	17	**10**	tseng	1047	9	**6**
diffeq	1497	10	**7**	pdc	4575	≥ 31	**16**	Total	--	≥ 331	**197**

Table 5 compares the number of tracks required to place and completely route circuits with VPR with the number required to place and globally route the circuits with VPR and then perform detailed routing with SEGA [23]. Table 5 also gives the size of each circuit, in terms of the number of logic blocks. The entries in the SEGA column with a ≥ sign could not be successfully routed because SEGA ran out of memory. Using SEGA to perform detailed routing on a global route generated by VPR increases the total number of tracks required to route the circuits by over 68% vs. having VPR perform the routing completely. Clearly SEGA has difficulty routing large circuits when input doglegs are not allowed.

To encourage other FPGA researchers to publish routing results using these larger benchmarks, we issue the following "FPGA challenge." Each time verified results which beat the previously best verified results on these benchmarks are announced, we will pay the authors $1 (sorry, $1 Cdn., not $1 U.S.) for each track by which they reduce the total number of tracks required from that of the previously best results. The technology-mapped netlists, the placements generated by VPR and the currently best routing track total are available at http://www.eecg.toronto.edu/~jayar/software.html.

6 Conclusions and Future Work

We have presented a new FPGA placement and routing tool that outperforms all such tools to which we can make direct comparisons. In addition we have presented benchmark results on much larger circuits than have typically been used to characterize academic FPGA place and route tools. We hope the next generation of FPGA CAD tools will be compared on the basis of these larger benchmarks, as they are a closer approximation of the kind of problems being mapped into today's FPGAs.

One of the main design goals for VPR was to keep the tool flexible enough to allow its use in many FPGA architectural studies. We are currently working on several improvements to VPR to further increase its utility in FPGA architecture research. In the near future VPR will support buffered and segmented routing structures, and soon after that we plan to add a timing analyzer and timing-driven routing.

References

[1] S. Brown, R. Francis, J. Rose, and Z. Vranesic, *Field-Programmable Gate Arrays*, Kluwer Academic Publishers, 1992.

[2] Xilinx Inc., *The Programmable Logic Data Book*, 1994.

[3] AT & T Inc., *ORCA Datasheet*, 1994.

[4] Actel Inc., *FPGA Data Book*, 1994.

[5] Altera Inc., *Data Book*, 1996.

[6] V. Betz and J. Rose, "Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size," *CICC*, 1997, pp. 551 - 554.

[7] S. Kirkpatrick, C. D. Gelatt, Jr., and M. P. Vecchi, "Optimization by Simulated Annealing," *Science*, May 13, 1983, pp. 671 - 680.

[8] V. Betz and J. Rose, "Directional Bias and Non-Uniformity in FPGA Global Routing Architectures," *ICCAD*, 1996, pp. 652 - 659.

[9] V. Betz and J. Rose, "On Biased and Non-Uniform Global Routing Architectures and CAD Tools for FPGAs," *CSRI Tech. Rep. #358*, Dept. of ECE, University of Toronto, 1996.

[10] C. E. Cheng, "RISA: Accurate and Efficient Placement Routability Modeling," *DAC*, 1994, pp. 690 - 695.

[11] M. Huang, F. Romeo, and A. Sangiovanni-Vincentelli, "An Efficient General Cooling Schedule for Simulated Annealing," *ICCAD*, 1986, pp. 381 - 384.

[12] W. Swartz and C. Sechen, "New Algorithms for the Placement and Routing of Macro Cells," *ICCAD*, 1990, pp. 336 - 339.

[13] J. Lam and J. Delosme, "Performance of a New Annealing Schedule," *DAC*, 1988, pp. 306 - 311.

[14] C. Ebeling, L. McMurchie, S. A. Hauck and S. Burns, "Placement and Routing Tools for the Triptych FPGA," *IEEE Trans. on VLSI*, Dec. 1995, pp. 473 - 482.

[15] C. Y. Lee, "An Algorithm for Path Connections and its Applications, "*IRE Trans. Electron. Comput.*, Vol. EC=10, 1961, pp. 346 - 365.

[16] J. S. Rose, W. M. Snelgrove, Z. G. Vranesic, "ALTOR: An Automatic Standard Cell Layout Program," *Canadian Conf. on VLSI*, 1985, pp. 169 - 173.

[17] J. S. Rose, "Parallel Global Routing for Standard Cells," *IEEE Trans. on CAD*, Oct. 1990, pp. 1085 - 1095.

[18] S. Brown, J. Rose, Z. G. Vranesic, "A Detailed Router for Field-Programmable Gate Arrays," *IEEE Trans. on CAD*, May 1992, pp. 620 - 628.

[19] G. Lemieux, S. Brown, "A Detailed Router for Allocating Wire Segments in FPGAs," *ACM/SIGDA Physical Design Workshop*, 1993, pp. 215 - 226.

[20] Y.-L. Wu, M. Marek-Sadowska, "An Efficient Router for 2-D Field-Programmable Gate Arrays," *EDAC*, 1994, pp. 412 - 416.

[21] Y.-L. Wu, M. Marek-Sadowska, "Orthogonal Greedy Coupling -- A New Optimization Approach to 2-D FPGA Routing," *DAC*, 1995, pp. 568 - 573.

[22] M. J. Alexander, G. Robins, "New Performance-Driven FPGA Routing Algorithms," *DAC*, 1995, pp. 562 - 567.

[23] G. Lemieux, S. Brown, D. Vranesic, "On Two-Step Routing for FPGAs," *Int. Symp. on Physical Design*, 1997, pp. 60 - 66.

[24] Y.-S. Lee, A. Wu, "A Performance and Routability Driven Router for FPGAs Considering Path Delays," *DAC*, 1995, pp. 557 - 561.

[25] M. J. Alexander, J. P. Cohoon, J. L. Ganley, G. Robins, "Performance-Oriented Placement and Routing for Field-Programmable Gate Arrays," *EDAC*, 1995, pp. 80 - 85.

[26] S. Wilton, "Architectures and Algorithms for Field-Programmable Gate Arrays with Embedded Memories," *Ph.D. Dissertation*, University of Toronto, 1997.

[27] S. Yang, "Logic Synthesis and Optimization Benchmarks, Version 3.0," *Tech. Report*, Microelectronics Centre of North Carolina, 1991.

[28] J. Cong and Y. Ding, "Flowmap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs," *IEEE Trans. on CAD*, Jan. 1994, pp. 1 - 12.