

# L4: AN FPGA-BASED ACCELERATOR FOR DETAILED MAZE ROUTING

*John A. Nestor, Jeremy Lavine*

ECE Department  
Lafayette College  
Easton, Pennsylvania USA  
email: nestorj@lafayette.edu

## ABSTRACT

This paper describes an FPGA-based accelerator for maze routing applications such as integrated circuit detailed routing. The accelerator efficiently supports multiple layers, multi-terminal nets, and rip up and reroute. By time-multiplexing multiple layers over a two-dimensional array of processing elements, this approach can support multi-layer grids large enough for detailed routing while providing at 1-2 orders of magnitude speedup over software running on a modern desktop computer. The current implementation supports a 32 X 32 routing grid with up to 16 layers in a single Xilinx XC2V6000 FPGA. Up to 64 X 64 routing grids are feasible in larger commercially available FPGAs. Performance measurements (including interface overhead) show a speedup of 29X-93X over the classic Lee Algorithm and 5X-19X over the A\* Algorithm. An improved interface design could yield significantly larger speedups.

## 1. INTRODUCTION

Routing is a key part of the physical design of electronic systems. While many different algorithms are used for routing, the classic Lee Algorithm for maze routing [1] remains a popular approach because it is guaranteed to find a shortest-path connection between two terminals.

The Lee Algorithm models the routing problem as a grid. It finds a connection between a source and destination terminal in three phases: 1) *expansion*, where gridpoints are labeled breadth-first in order of increasing distance from the source until the destination is encountered; 2) *backtrace*, where gridpoints are selected for a connection by following labels of decreasing distance from the target back to the source and marking the selected gridpoints as obstacles, and 3) *cleanup*, where expansion labels are removed.

The biggest disadvantage of the Lee Algorithm is that it is computationally expensive. The expansion phase has a worst case execution time of  $O(d^2)$  for a connection of distance  $d$  on a single layer and  $O(Ld^2)$  for multiple layers when  $d \gg L$ . The cleanup phase has an execution time of  $O(LN^2)$  for an  $L$ -layer  $N \times N$  grid. For this reason, routing problems are often subdivided into separate global and

detailed routing steps [2], with detailed routing focusing on subregions of the overall routing surface consisting of “a few tens of tracks in either direction” [3]. However, even on this reduced problem size routing consumes a large amount of execution time.

Because of this performance issue, several designs have been proposed for hardware acceleration of the Lee Algorithm. During expansion, all gridpoints at an equal distance from the source terminal can ideally be labeled in parallel, reducing the expansion time to  $O(d)$ . During cleanup, all gridpoints not representing routed wires can be cleared in one step, reducing cleanup time to  $O(1)$ .

The most straightforward approach to exploiting this parallelism is to create an array of small processing elements (PEs) that directly represent the points in the grid. Connections between neighboring PEs allow the expansion phase to proceed in parallel. This *full-grid* [4-6] approach has the best potential for speedup but comes at a high cost in hardware, since a total of  $LN^2$  PEs are required.

An alternative *virtual-grid* [7-9] approach uses a smaller array of PEs and maps multiple gridpoints onto each PE. This reduces the total number of PEs but complicates the design of each individual PE, which must now keep track of each gridpoint that it represents and communicate that information with neighboring PEs. Other approaches have proposed the use of systolic arrays [10] or special-purpose hardware that accelerates a portion of the algorithm without addressing its full potential parallelism [11].

More recent work on routing acceleration has focused on FPGAs. In FPGA routing the grid graph is replaced by a directed graph that models the FPGA’s programmable interconnect. For example, [12] reported an accelerator for the well-known PathFinder [13] FPGA routing algorithm. This approach combined an FPGA-based priority queue implementation with a network of workstations which took advantage of coarse-grained parallelism in the algorithm for a reported 15X speedup over contemporary (1997) workstations.

Other work [14] proposed an aggressive acceleration approach in which the interconnection structure of a Fat-Tree topology FPGA was augmented with hardware to directly support connection search, allocation of routed connections, and ripup (“victimization”) of routed connections when no connection can be found. Simulations

of this design predict a speedup of up to three orders of magnitude over PathFinder with design quality within 5%-25% of software. The estimated hardware overhead varies from 1.5X for support just of path search to 2.5X when path allocation and victimization are included. Further work [15] added support for multiple-terminal nets, improved routing quality by considering congestion, added support for mesh-style routing topologies, and explored implementing the routing hardware on an existing FPGA. While simulations predict dramatic speedups, the hardware cost is high – each PE requires an estimated 155 4-input lookup tables (LUTs) for a single-layer mesh topology.

In contrast to these approaches, L3 [16] focuses on the classic multi-layer maze routing. It is intended for detailed routing of integrated circuits rather than FPGAs. L3 uses a modified full-grid approach with a two-dimensional array of PEs that supports multi-layer routing. Each PE stores the state of all layers at a particular  $(x, y)$  coordinate and time-multiplexes each layer's gridpoint  $(x, y, 0), (x, y, 1) \dots (x, y, L-1)$  through a common state sequencer. This allows efficient processing of multi-layer grids using compact hardware.

L3 showed the potential of this approach, but was implemented in a relatively small FPGA that supported a 4-layer 16 X 16 grid. Moreover, L3 supported routing only two-terminal nets and provided limited support for rip up and reroute. Finally, the prototype design lacked a high speed interface to a host PC, and speedups were measured without considering the interface overhead.

This paper describes an improved accelerator design called L4 that builds on the L3 approach. Like L3, it uses a two-dimensional grid and time-multiplexes this grid over multiple layers. However, it supports larger arrays and interfaces to a host computer using a PCI interface.

L4 directly supports the routing of multiterminal nets. In addition, a new feature called *etching* is provided to speed rip up and reroute when routing fails due to congestion. Etching provides a way to complete the connection of a blocked net while attempting to minimize the number of nets that must be ripped up.

A complete L4 prototype has been implemented on a Xilinx XC2V6000 FPGA that supports a 16-layer 32 X 32 grid. The FPGA is interfaced to a host PC to provide a complete system for routing under software control. This physical prototype is in contrast to many previous approaches which present results based on simulation only. Measurements that *include* the overhead of a relatively primitive PCI interface show significant speedups.

The rest of this paper is organized as follows: Section 2 describes the organization of the L4 architecture. Section 3 describes how L4 supports routing of multi-terminal nets. Section 4 introduces the concept of etching for use in ripup and reroute. Section 5 describes the L4 implementation and presents performance results. Section 6 concludes the paper and provides suggestions for future work.

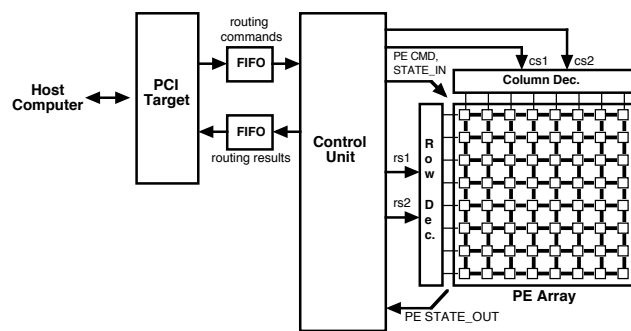


Fig. 1. L4 Organization.

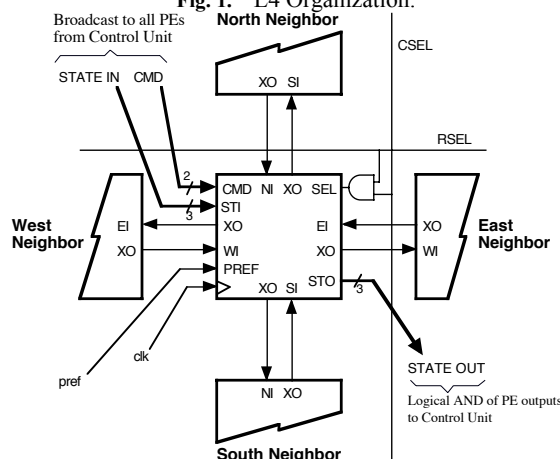


Fig. 2. PE Connection Detail.

## 2. L4 ORGANIZATION

Fig. 1 shows the general organization of the accelerator. The heart of the design is a two-dimensional array of PEs attached to a control unit. Each PE corresponds to a horizontal  $(x, y)$  position on the routing grid and represents the gridpoints on every layer at that horizontal position.

Fig. 2 shows how a PE is connected to neighboring PEs for local communication during expansion. PEs can also be selected for specific operations using a row decoder and column decoder. The decoders can select either a single row/column or a range of rows and columns. This allows the selection of either a single PE or an arbitrary rectangle of PEs for parallel operation.

Every PE has CMD and STATE\_IN inputs that are broadcast from the control unit to all PEs simultaneously. It also has a STATE\_OUT output; the control unit reads the logical AND of this output from all PEs.

As gridpoints from each successive layer circulate through the state sequencer of each PE from bottom to top, the function of the PE and next state (NS) of the current gridpoint are determined by the current state (CS), the CMD input and row/column select inputs RSEL and CSEL.

Table 1 summarizes the commands performed by each PE. The READ and WRITE commands are similar to read and write operations in a traditional memory.

**Table 1.** PE Commands.

Command	Action
READ	if (RSEL & CSEL) STATE_OUT = CS
WRITE	if (RSEL & CSEL) NS = STATE_IN
CLEARX	if CS = (XE, XW, ..., XU, XD) NS=EMPTY
EXPAND	if (EI) NS = XE else if (WI) NS = XW; else if (NI) NS = XN else if (SI) NS = XS; else if (UI) NS = XU; else if (DI) NS = XD

**Table 2.** Control Unit Commands.

Command	Action
ROUTE $x1\ y1\ z1\ x2\ y2\ z2$	Route a 2-terminal connection
SELECT $x1\ y1\ z1\ x2\ y2\ z2$	Select a rectangular region
READ	Read state of selected region
WRITE <i>state</i>	Write <i>state</i> to selected region

However, since multiple PEs can be selected for an operation, WRITE can be used to set the state of all PEs in an arbitrary rectangular region in parallel. This is useful for initializing an obstacle, ripping up a rectangular wire segment, and clearing all gridpoints on the current layer. Similarly, the READ operation can read all gridpoints in a rectangular region and deliver the logical AND of these results to the controller. This can be used as a form of "line probe" that determines whether all selected gridpoints are in the EMPTY state (i.e., they are free for a new connection).

The CLEARX command is used during the cleanup phase to return "expanded" gridpoints to the EMPTY state. This operates to perform cleanup in L clock cycles.

The EXPAND command is the key function of the PE because it accelerates the  $O(Ld^2)$  expansion phase to  $O(Ld)$ . When EXPAND is applied, every PE monitors the expansion status of neighboring gridpoints, including gridpoints immediately above and below the current gridpoint (shown as UI, DI and available within the PE). When a neighboring gridpoint is in an expansion state, the current gridpoint enters an expanded state (i.e., XE, XW, XN, XS, XU, or XD) that indicates the direction from which the expansion arrived (and for the backtrace phase, the direction of the shortest path back to the source).

During EXPAND every PE provides status outputs on two bits of STATE\_OUT; the logical AND of these outputs is monitored by the control unit. STATE\_OUT[0] is asserted low by a selected PE if it is in an expansion state. The control unit uses this signal to determine when a connection target is reached. STATE\_OUT[1] is asserted low by any PE that is currently entering an expanded state. The control unit uses this signal as a watchdog since when high it indicates that no further expansion is possible.

The control unit is a finite state machine that accepts high-level routing commands from a host processor and broadcasts low-level commands to the PE array. Table 2 summarizes the commands implemented by the control unit. Each command is a 32-bit word that is passed from

the host processor to the control unit using a FIFO. 32-bit result words are passed back to the host processor in the form of endpoints of wire segments and status words indicating either successful completion of the command or failure.

### 3. MULTI-TERMINAL NETS

The basic Lee Algorithm supports only two-terminal nets. However, multi-terminal nets can be routed as follows [2]: First, two terminals are routed, resulting in a connection between the two terminals. Next, the gridpoints that represent this connection are relabeled as source nodes and used to start an expansion search to the third terminal. This process is repeated until all terminals are connected.

The L4 accelerator implements this approach with an additional gridpoint state called TRACED. During multi-terminal routing, the backtrace phase labels connections with the TRACED state instead of the BLOCKED state. During subsequent expansions, cells in the TRACED state are treated as "expanded" states by neighboring PEs, allowing expansion to proceed from all gridpoints in the partial connection. When all terminals in the net are connected, a final cleanup phase returns all gridpoints that are in the TRACED state to the BLOCKED state in parallel.

The control unit implements two new commands to support multi-terminal routing: ROUTE\_EXTEND\_INIT, which initializes multi-terminal routing and routes the first two terminals, and ROUTE\_EXTEND, which is used to add connections to each additional terminal.

### 4. RIPUP AND REROUTE SUPPORT

A major drawback of maze routing is that it only considers one connection at a time. A shortest path connection for one net can block connections for nets that have not yet been routed. To overcome this problem, nets are usually routed in varying order using a rip up and reroute strategy [2]. In this approach, nets are routed first in an arbitrary order. When a net cannot be connected, "blocking" nets that prevent this connection must be removed. The blocked net is then routed followed by other nets.

L4 supports net ripup using the ability to select and write a rectangular region simultaneously. This allows each horizontal (east/west or north/south) segment to be removed by the controller using one WRITE command for each segment (instead of one command for each gridpoint).

A second and more important problem involves the identification of nets which must be removed to complete a blocked connection. The expansion search implemented by L4 cannot help here, since when it fails it only indicates that a path cannot be found.

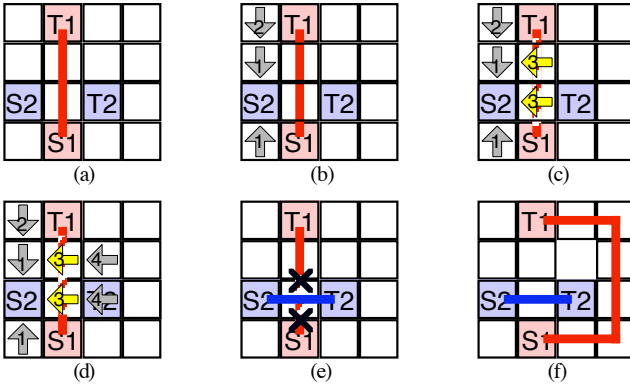


Fig. 3. Etching – support for rip up and reroute.

Some software routers (e.g., [17]) attack this problem using a penalty function where expansion is allowed through gridpoints containing prior connections at a much higher cost than through empty gridpoints. If no obstacle-free path is found, the minimum cost path identifies the minimum number of occupied gridpoints that must be ripped up to complete the routing of the current net.

Supporting a penalty function in a software implementation requires a priority queue. In a full-grid accelerator, the penalty function can be realized by adding a counter to each PE [5]. This is not feasible in L4 because each PE represents multiple layers; the cost of counter storage for each layer would be excessive.

Instead, L4 uses an alternative approach called *etching* [18]. The idea of etching is analogous to the use of a solvent to remove a physical barrier in a manufacturing process. Etching is performed by each PE under control of an "etch enable" signal from the control unit. When etching is disabled, the PE operates as before. When etching is enabled, each PE allows expansion to occur through a gridpoint representing an obstacle as well as an empty gridpoint. The control unit normally operates with etching disabled, but if expansion fails, it enables etching for one expansion cycle, allowing expansion through routed gridpoints immediately adjacent to the limits of the previous expansion. It then continues the search with etching disabled again. If routing fails again, etching is again enabled for an additional expansion cycle.

While etching can be useful for removing wires that block a connection, it cannot be applied indiscriminately – connection terminals must not be removed, and routing problems may contain objects which cannot be removed. These gridpoints are marked with a new UNETCHABLE state, which cannot be removed by etching.

Fig. 3 illustrates the application of etching on a single-layer 4 X 4 grid with two nets. The terminals of each net are set to the UNETCHABLE state so that they cannot be removed during etching. Net 1 from source S1 to target T1 is routed first; this blocks the connection for Net 2 (a). During normal expansion of Net 2, all gridpoints that can be reached from source terminal S2 are labeled before

expansion fails after two expansion steps (b). At this point, the control unit enables etching for the third expansion step, expanding two obstacle gridpoints representing the current wire for Net 1 (c). Normal expansion then resumes and the target T2 is reached in the fourth expansion step (d). During backtrace, the control unit marks gridpoints on the backtrace path as obstacles. It also reports etched gridpoints to the host processor, which must then identify which nets have been cut using an intersection check with previously routed nets. During cleanup, all normally expanded gridpoints are returned to the EMPTY state (e). Etched cells, on the other hand, are returned to the BLOCKED state. Finally, any cut nets must be completely ripped up and rerouted (f).

The PE implementation supports etching with a new ETCH\_ENABLE control input and an added state bit ETCHED that represents whether a cell was originally empty when labeled (false) or originally an obstacle (true). During backtrace, this bit identifies gridpoints that must be reported to the host processor as etched. During cleanup, the ETCHED bit allows gridpoints to be simultaneously returned to the EMPTY or BLOCKED state as appropriate.

## 5. IMPLEMENTATION AND RESULTS

The L4 implementation is coded in Verilog. The basic PE design consists of 116-147 lines of Verilog depending on which features are used. When compiled into hardware with the Xilinx ISE synthesis tool, the basic PE without etching or multiterminal routing requires 38 Look-Up Tables (LUTs) and 4 flip-flops. A PE that implements etching as well as the basic function requires 49 LUTs and 5 flip-flops. A PE design that includes both etching and multiterminal routing requires 62 LUTs and 11 flip-flops.

Table 3 provides implementation details for a complete 32 X 32 routing array that supports etching and multiterminal routing for up to 16 layers. It is implemented using a Xilinx Virtex-II XC2V6000 FPGA [19] on a Dini Group 3000K10S [20] development board. The board includes a PCI interface that plugs into to a standard PCI slot on a desktop PC host.

The clock rate of the accelerator is limited by the delay path between the control unit and the routing array. To increase the clock rate over the previous L3 design, registers are included in the row and column decoders and the status output of the array. Using a relatively low (-4) speed grade FPGA, the routing array can operate at a frequency of 30 MHz, a significant improvement over the 24 MHz L3 accelerator despite the fact that the L4 array is four times larger than the 16 X 16 L3 array.

The current host is a 1.79Ghz Pentium 4 desktop PC running RedHat Linux 7.2. Host software sends routing commands and receives routing results using I/O ports that are memory mapped through a Linux device driver.

**Table 3.** Implementation Results.

Component	Lines Src.	LUTs	FFs
PCI Target (from Dini)	529	164	223
CMD FIFO (Xilinx IP)	IP	78	126
Result FIFO (Xilinx IP)	IP	78	126
PE Array	749	64,838	11,268
Column Decoder	94	211	32
Row Decoder	94	211	32
Control Unit	1,103	362	129
FPGA Top Level	272	38	39
<b>TOTAL</b>	<b>2,841</b>	<b>65,980</b>	<b>11,975</b>

**Table 4.** Routing Speedup

Example	Lee	A*	L4	Spdup	
	( $\mu$ s)	( $\mu$ s)	( $\mu$ s)	Lee	A*
adjnt6	62.90	91.78	12.29	5.12	7.47
adjnt8	82.49	112.05	12.35	6.68	9.07
adjnt16	162.20	192.63	13.38	12.12	14.39
corner6	2,170.52	158.2	32.58	66.62	4.85
corner8	2,905.91	180.97	37.71	77.06	4.79
corner16	5,849.52	271.35	66.77	87.61	4.06
t6_90	85,585.41	N/A	2,964.37	28.87	N/A
t8_90	119,321.59	15,337.19	3,110.83	38.36	4.93
t16_90	253,031.40	22,210.67	3,545.12	71.37	6.27
m6_40	108,112.72	19,569.89	2,932.49	36.87	6.67
n8_40	150,573.09	19,992.06	3,050.88	49.35	6.55
m16_40	324,452.70	28,653.52	3,468.57	93.54	8.26

**Table 5.** Etching Experiment

System	Segments	Wirelen.	EtchPts	Time	Spdup
				( $\mu$ s)	
L4	1,862	5,142	40	9,562	-
Lee w/P	1,745	4,829	93	749,628	78X
A* w/P	1,888	4,839	148	186,015	19X

Table 4 summarizes performance measurements that compare the performance of L4 to software implementations of the Lee Algorithm and A\* [23] Algorithm that are written in C. Hardware measurements were performed using the configuration described in the previous paragraph. Software measurements were performed using a lightly loaded desktop PC with a 3.79 GHz EMT64 Pentium 4 and 3.5GB of RAM running 64-bit Ubuntu Linux v5.05. Both sets of measurements were performed using the Pentium 4 cycle counter to measure the elapsed time between the start and end of each command and the equivalent software.

Measurements were performed for accelerators instantiated to support 6, 8, and 16 layer grids on four sets of benchmarks. The first two sets of benchmarks are simple problems that are intended to show the range of

speedups between a worst case (routing between adjacent gridpoints (0,0,0) and (0,0,1), which requires almost no expansion and a best case (routing between the grid “corners” at (0,0,0) and (31,31,L-1), which requires expansion of the entire grid by the Lee Algorithm. The third set measures routing time for 90 successive randomly selected two-terminal nets, while the fourth set measures 40 successive randomly selected multi-terminal nets, each with 2-5 terminals. Routing quality was comparable for each approach (total wirelength varied by less than 2.5%).

L4’s speedup compared to the classic Lee Algorithm is dramatic (5X-94X), but the speedup over the A\* algorithm is lower (4X-14X) because the A\* algorithm biases its search toward the target and reduces the cost of the expansion phase. However, even the worst case shows a 4X speedup due to acceleration of the cleanup phase.

Much of the accelerator’s time is consumed by interface overhead. To quantify this, a separate cycle counter was added to the accelerator hardware that counts accelerator clock cycles from beginning to end of each routing command. Subtracting this time from the time measured in the host processor gives the interface overhead. On the 6-layer 40 random multi-terminal net problem, interface overhead per net varies between 12.5 $\mu$ s and 117.2 $\mu$ s, with an average of 56.2 $\mu$ s (75% of the average routing time). The wide variation is due to the fact that routes containing several segments require a separate word transfer for each segment. Using block transfers could greatly improve performance, as would replacing the PCI interface with a faster interface.

These accelerator clock measurements also allow us to estimate an upper bound on the best-case speedup that could be achieved if interface overhead could somehow be completely removed. On individual nets, speedup ranges from 4.6X – 240X over the Lee Algorithm and 12X-43X over the A\* algorithm. While some interface overhead is inevitable, a better design could significantly improve performance.

A final experiment was performed to evaluate the effectiveness of etching compared to software routing with a penalty function. In this experiment, a set of 90 random multi-terminal nets were generated for an 8 layer 32 X 32 grid. The resulting problem is too congested to route completely, but can be used to evaluate how different routing algorithms select gridpoints for ripup.

L4 was compared to both the classic Lee Algorithm and the A\* algorithm with a cost penalty for crossing obstacles. This result of this experiment is equivalent to one pass of a ripup-and-reroute algorithm in which “violations” have occurred but will be corrected in a later pass.

Table 5 summarizes the results in terms of number of rectilinear wire segments in the final routing, total wire length, the number of gridpoints selected for ripup during etching (“EtchPts”), and the required execution time and the speedup of L4 compared to the software approaches.

The “EtchPts” metric is important because it identifies the number of gridpoints assigned to more than one net; these nets must be identified and ripped up in a later pass. A smaller number here implies less work to be done during the ripup stage. The hardware etching approach results in a significantly smaller number of ripped up gridpoints, but does so at an expense of about 6.5% overall wirelength. The segment count is significant because it corresponds to the number of bends found in each connect. L4 falls between the Lee and A\* approaches on this metric. Finally, L4 displays higher speedups during etching for both the Lee and A\* algorithms compared to Table 4; this is because all gridpoints that are reachable from the source gridpoint must be expanded before obstacle gridpoints can be considered.

## 6. CONCLUSION

This paper has described the design of an FPGA-based hardware accelerator for maze routing that is intended for detailed routing of integrated circuits. Experiments show promising speedups over software implementations that could be further enhanced by an improved interface design.

The 32 X 32 X 16 grid size approaches the size needed to support detailed routing, and larger commercially available FPGAs could support even larger arrays. For example, the Xilinx Virtex-4 XC4VLX200 could support up to 64 X 64 X 16 grids for basic routing and 50 X 50 X 16 grids for the full design described in this paper.

There are a number of areas for future work in this project. First, the basic net routing capability here must be exploited by developing a ripup and reroute algorithm that uses the routing accelerator to search for multiple connections. This effort is currently underway. Second, modern routing tools must deal many issues that go beyond simple path search, such as non-uniform spacing, preferential routing directions on different layers, via restrictions, and other physical considerations and electrical considerations such as cross-talk and signal integrity. Further research must address these issues. Finally, the L4 approach could be extended to FPGA routing by modifying the PE design to represent FPGA routing resources.

## 7. REFERENCES

- [1] C.Y. Lee, "An Algorithm for Path Connections and its Applications", *IRE Transactions on Electronic Computers*, Vol. EC-10, 1961, pp. 346-365.
- [2] N. Sherwani, *Algorithms for VLSI Physical Design Automation*, 3rd ed., Kluwer Academic Press, 1999.
- [3] S. Batterywala, et. al., "Track Assignment: A Desirable Intermediate Step Between Global Routing and Detailed Routing", *Proc. ICCAD*, November 2002, pp. 59-66.
- [4] M. Breuer and K. Shamsa, "A Hardware Router", *Journal of Digital Systems*, Vol. IV, Issue 4, 1981, pp. 393-408.
- [5] A. Iosupovici, "A Class of Array Architectures for Hardware Grid Routers", *IEEE Transactions on CAD*, Vol. CAD-5, April 1986, pp. 245-255.
- [6] T. Ryan, and E. Rogers, "An ISMA Lee Router Accelerator", *IEEE Design & Test of Computers*, Vol 4, Oct. 1987, pp. 38-45.
- [7] K. Suzuki, et. al., "A Hardware Maze Router with Application to Interactive Rip-Up and Reroute", *IEEE Transactions on CAD*, Vol. CAD-5, Oct. 1986, pp. 466-476.
- [8] T. Watanabe, T., et. al., "A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor", *IEEE Transactions on CAD*, Vol. CAD-6, Mar. 1987, pp. 241-249.
- [9] R. Ventkateswaran, and P. Mazumder, "A Hexagonal Array Machine for Multilayer Wire Routing", *IEEE Transactions on CAD*, Vol. 9, Oct. 1990 pp. 1096-1112.
- [10] R. Rutenbar and D. Atkins, "A Class of Cellular Architectures to Support Physical Design Automation", *IEEE Transactions on CAD*, CAD-4 (1984) 264-278.
- [11] Won. Y, Sahni, S., and El-Ziq, Y., "A Hardware Accelerator for Maze Routing", *Proceedings Design Automation Conference*, 1987, 800-806.
- [12] P. Chan and M. Schlag, "Acceleration of an FPGA Router", in: *Proceedings 5th Annual Symposium on FPGAs for Custom Computing Machines*, 1997, 175-181.
- [13] L. McMurchie and C. Ebeling, "Pathfinder: A Negotiation-Based Performance-Driven Router for FPGAs", *Proceedings 3rd International Symposium on FPGAs*, pp. 140-149, 1995.
- [14] A. DeHon, R. Huang, and J. Wawrzynek, "Hardware Assisted Fast Routing", *Proceedings IEEE Symposium on FPGAs for Custom Computing Machines*, 2002 205-215.
- [15] R. Huang, J. Wawrzynek, and A. DeHon, "Stochastic, Spatial Routing for Hypergraphs, Trees, and Meshes", *Proc. International Symposium on FPGAs*, 2003, 78-87.
- [16] J. A. Nestor, "L3: An FPGA-Based Multilayer Maze Routing Accelerator", *Microprocessors and Microsystems*, Vol. 29, No. 2-3, pp. 87-97, March 2005.
- [17] Y. Lin, Y. Hsu, and F. Tsai, "SILK: A Simulated Evolution Router", *IEEE Transactions on CAD*, Vol. 8., No. 10, pp. 1108-1114, October 1989.
- [18] K. Nasabzadeh, "Extensions to Direct Grid Maze Routing Accelerators". BS Honors Thesis, Lafayette College, May 2005.
- [19] Xilinx, Corporation, *Xilinx Databook*, 2006, <http://www.xilinx.com>.
- [20] The Dini Group, *DN3000k10S User's Manual Version 1.2*, April, 2003. Available at <http://www.dinigroup.com>.
- [21] F. Rubin, "The Lee Path Connection Algorithm", *IEEE Trans. Computers*, Vol. C-23, No. 9, pp. 907-914, Sept. 1974.