

# FPGA Implementation of a Multilayer Maze Router

J.A. Nestor<sup>1</sup>

<sup>1</sup>Department of Electrical and Computer Engineering, Lafayette College, Easton, PA 18042

## Abstract

This paper describes the design and implementation of an FPGA-based multilayer maze routing accelerator. The accelerator is implemented as an array of small processing elements that corresponds to the horizontal structure of the routing grid. Multilayer routing is efficiently implemented by time-multiplexing multiple layers over the two-dimensional grid. Prototype accelerators supporting 8 X 8 X 4 grids have been implemented and tested in a low-end Xilinx FPGA, and larger accelerators are now under development. Preliminary performance data shows speedups of 50-75 over software for an 8 X 8 X 4 grid; higher speedups are expected for larger grids.

## I. INTRODUCTION

The problem of *maze routing* involves finding a shortest-path connection on a grid with obstacles. It has been applied extensively to VLSI wire routing [1] but is also useful in other applications such as robot path planning. While many different approaches to maze routing have been proposed, the classic Lee Algorithm [2] remains popular because it is guaranteed to find a shortest-path connection between a *source* and target *terminal* if one exists.

The Lee Algorithm represents the routing surface as a rectangular grid of connected nodes. The algorithm operates in three phases, as shown in Figure 1. During the *expansion* phase, it searches outward from the source while labeling each encountered node with the direction of the shortest path back to the source. Expansion continues until either the target is reached or no further labeling is possible, in which case no connection exists. During the *backtrace* phase, the algorithm follows the labels from the target node back to the source node and marks the nodes on these paths as *obstacles*, indicating that these nodes are part of a connection and unavailable for further routing. During the *cleanup* phase (not shown), it clears the remaining labels so that another connection can be routed. The algorithm is easily extended to multiple layers using a three-dimensional grid.

Although popular because of its shortest-path guarantee, the Lee Algorithm is also computationally expensive- the expansion phase is  $O(d^2)$  for a connection of distance  $d$ , while the cleanup phase is  $O(N^2)$  for an  $N \times N$  grid. Multilayer routing is even worse since expansion can proceed on up to  $L$  layers simultaneously.

The high computational cost of the Lee Algorithm has motivated several proposals for hardware accelerators that use some form of parallelism to speed up the routing process. One of the earliest and most straightforward approaches was the *full grid* or *direct grid* approach [3-5], in which points on the routing grid are mapped to an array of simple processing elements (PEs). Each PE corresponds to one point on the

grid, and each PE connects to PEs corresponding to neighboring gridpoints. During expansion, PEs communicate with adjacent PEs so that all nodes at the same distance from the source node are labeled simultaneously. This reduces the complexity of expansion to  $O(d)$  complexity; cleanup becomes a simple reset operation that can be performed in constant time. However, this approach is expensive in terms of hardware, since  $N^2$  processing elements are required to implement a single-layer grid and  $L \times N^2$  processing elements for multilayer grid.

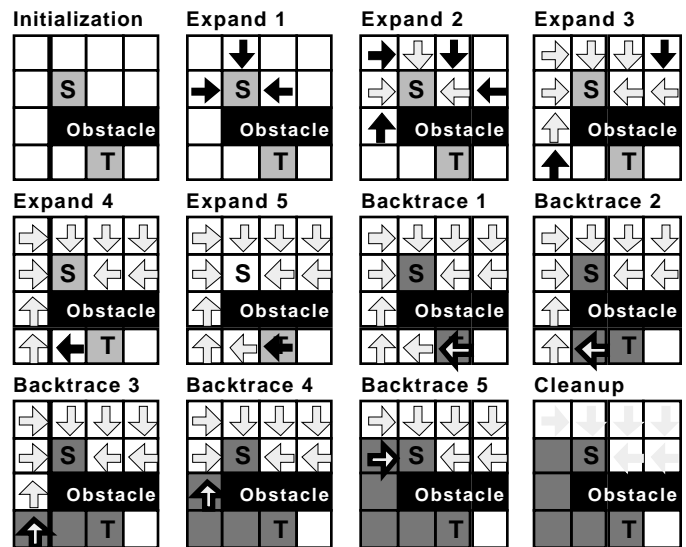


Figure 1: Maze Routing with the Lee Algorithm. In software, nodes are labeled sequentially during the expansion phase. In a full-grid accelerator, all nodes at the same distance from the source are labeled in parallel.

An alternative *virtual grid* approach [6-8] maps multiple gridpoints to each processing element. This reduces the size of the array at the cost of increased execution time and significantly more complex processing elements.

Other approaches used raster pipelines and other special-purpose architectures [9-11] which have lower hardware costs but also correspondingly reduced acceleration.

More recently, routing accelerators have been proposed for routing FPGAs, which have a more constrained routing topology. These include an approach that accelerates the priority queue of an existing algorithm [12,13] as well as a more radical approach that builds routing capability into the FPGA itself [14,15].

This paper describes modified full-grid approach to multilayer routing called L3 [16, 17] and its implementation in an FPGA. The key idea in this approach is to time-multiplex a two-dimensional array of PEs over multiple layers. The state of each PE is stored in a shift register that circulates the state

of each layer through a common sequencing unit. Since Xilinx FPGAs allow up to 16-bit shift registers to be implemented in one look-up-table (LUT) [18], this provides economical support for up to 16 layers.

Time-multiplexing thus makes multiple layer routing feasible but increases the time complexity to  $O(d \times L)$ . However, since the number of layers is typically much smaller than the overall grid size, this is a reasonable tradeoff. Moreover, if preferential routing is used where horizontal connections are preferred over vertical then this penalty can be significantly reduced.

Additional improvements in L3 over previous full-grid approaches include a significant reduction in the logic required for each PE, support for rapid removal of obstacles and connections as required by routers using a “rip-up-and-reroute” strategy, and support for preferential routing on a layer-by-layer basis. The grid-based nature of the L3 architecture maps naturally to the structure of common FPGAs, while the local nature of most connections allows high utilization of logic blocks. As larger FPGAs come to market, larger arrays can be implemented with only minor modifications to the architecture.

This paper is organized as follows: Section 2 describes the general organization of the L3 architecture. Section 3 describes how the architecture is used to route single-layer connections, while Section 4 describes how the architecture is extended to perform multi-layer routing using time-multiplexing. Section 5 describes the design of the control unit that sequences the operation of the processing elements. Section 6 describes implementation results. Finally, Section 7 concludes the paper and discusses possible future work.

## II. GENERAL APPROACH

To design an effective hardware accelerator for maze routing, it is important to understand how such algorithms are used in modern routing tools.

First, these routing tools must wire a large number of connections or *nets*. Since the Lee algorithm can only make one connection at a time, it is used *iteratively*. While the Lee algorithm is guaranteed to find a shortest path for a single net, each routed net forms obstacles for later nets. These obstacles may make the routing of later nets longer than optimal, and sometimes impossible to complete. This requires a “rip-up-and-reroute” strategy in which the connections for some nets are removed and re-routed in a different order in an attempt to improve the routing.

Second, modern design technologies require the use of several layers of interconnect. Current VLSI processes provide up to 8 layers, while multilayer printed circuit boards may use more than 30 layers. Thus to be effective, any attempt to accelerate maze routing must accommodate multiple layers efficiently. Some of these layers may have preferred directions for routing (e.g. all horizontal or all vertical).

The L3 architecture was designed with these needs in mind. While based on early full-grid approaches there are some key differences: efficient support for multi-layer routing with preferred directions, efficient cell implementation, and support for rip-up-and-reroute.

Figure 2 shows the basic organization of the L3 architecture. Like other full-grid accelerators, it consists of an array of processing elements called *cells* and an attached control unit (not shown). Each cell is a finite state machine that represents the status of the gridpoint during routing.

Each cell is connected locally to neighboring cells in the grid using output XO. XO is true when a cell is labeled during expansion. It is connected to the WI, EI, NI, and SI inputs of the cell’s west, east, north, and south neighbors, respectively.

The attached control unit communicates with the array of cells using a global command bus CMD, and a row and column decoder that allow the selection of either individual cells or rectangular regions of cells. Some commands are single-instruction multiple-data (SIMD) commands that specify an action to be performed on all cells. Other commands apply only to cells selected using the row and column decoders. The STATUS signal connects all cells to the control unit via a tristate/wired-NOR bus.

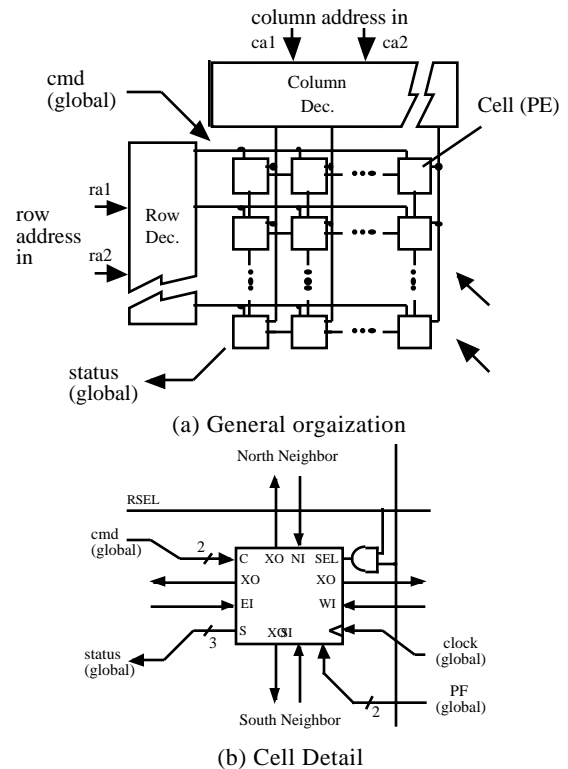


Figure 2: L3 Organization. An attached control unit broadcasts commands via the CMD signal while monitoring the STATUS bus. Cells are selected using the row and column decoders.

## III. SINGLE-LAYER OPERATION

In single-layer routing, each cell is a finite state machine with 6 states: E (empty), BL (blocked), XE (expanded east), XW (expanded west), XN (expanded north) and XS (expanded south). Each “expanded” state encodes the direction of the shortest path back to the source node.

Table 1 describes the function of each cell as it responds to one of four commands: CLEAR, SET, EXPAND, and TRACE.

Table 1. Cell state sequencing in response to CMD. "Any X" indicates any expanded state (XN, XS, XU, or XD).

CMD	SEL	EW * PF1	WI * PF1	NI * PF0	SI * PF0	PS	NS	S1	S0
<b>CLEAR</b>	0	–	–	–	–	Any X	E	1	1
	1	–	–	–	–	Any	E	1	1
<b>SET</b>	1	–	–	–	–	Any	XE	1	1
<b>TRACE</b>	1	–	–	–	–	Any	BL	D1	D0
<b>EXPAND</b>	–	1	–	–	–	E	XE	0	SEL'
	–	0	1	–	–	E	XW	0	SEL'
	–	0	0	1	–	E	XN	0	SEL'
	–	0	0	0	1	E	XS	0	SEL'
	–	–	–	–	–	Any X	PS	1	SEL'
All Other Conditions							PS	1	1

Single-layer routing starts when the control unit selects all cells and broadcasts a CLEAR command to set all cells to the E state. It then selects the source node and uses the SET command to set the source cell to the XE state. It next applies the EXPAND command while selecting the location of the target cell. During each successive clock cycle, a cell will enter an expanded state if one of its neighbors is in an expanded state and the corresponding preference input (PF1 for east/west, PF0 for north/south) is high. This allows preferential expansion to be performed by the control unit, e.g., to bias expansion (and connections) in either the east-west or north-south direction.

Expansion is terminated depending on the value of two status bits, labeled S1 and S0. Status bit S0 indicates whether the selected target cell has been reached during expansion – it is pulled low by the target cell when it enters an expansion state to indicate that expansion has successfully completed. Status bit S1 is a “watchdog” bit that is pulled low by any cell in the array that is currently entering an expansion state. A high value on S1 indicates that expansion has failed because obstacles block all possible connections between the source and target.

If expansion is successful, the control unit starts the backtrace phase by selecting the target cell and broadcasting the TRACE command. In response, the target cell asserts its state code on the STATUS bus and enters the obstacle state (BL). The control unit uses the direction information encoded in the state code to determine the address of the next cell in the path and repeats this process until the source node is reached and the entire path is marked as an obstacle.

Finally, during the cleanup phase the CLEAR command is applied with no cells selected. This clears expansion labels (but not obstacles) so that additional nets can be routed using the same process.

The entire process takes  $d$  clock cycles for expansion,  $d$  clock cycles for backtrace, and 1 clock cycle for cleanup. This is a substantial improvement over the  $O(d^2)$  worst-case performance of expansion and the  $O(N^2)$  performance of cleanup in a conventional software approach.

#### IV. MULTI-LAYER ROUTING

L3M is the multi-layer version of the L3 architecture. It uses the same array structure but time-multiplexes cell hardware over multiple layers.

Figure 3 shows the internal organization of a L3M cell., which uses a shift register to store the states of each layer. It processes states from bottom to top on each successive clock cycle. An attached layer counter keeps track of the current layer being processed and asserts the /TOP signal low when the top layer is being processed.

During each clock cycle, the state of the layer that is currently being processed is stored in the register ST0. State information for the remaining layers is stored in the shift register. At the end of the clock cycle, the new state of the cell is shifted into the top stage of the shift register and the state in the cell “above” the current cell is loaded into ST0 to be processed on the next clock cycle.

The state sequencer is similar to that of the single-layer cell except that it has two additional states XU and XD. These indicate that the shortest path back to the source is either “up” or “down”, respectively, from the current node. As in the original cell it uses the existing EI, WI, NI, and SI inputs to determine the expansion status of its horizontal neighbors on a given layer. Additional logic is needed to determine the expansion status of the vertical neighbors i.e. the nodes immediately above and below the node being processed.

The logic block marked XH in Figure 2 determines the expansion status of the node above the current node. Its inputs are tied to the bottom stage of the shift register, which (unless the top layer is being processed) contains the state of the node immediately above the current node. This block asserts its output true when the node above the current node is in an expansion state. The AND gate connected to the XH block output suppresses this value when the /TOP signal is low; this indicates that the top layer is being processed and there is no layer above the top layer. This prevents expansion

from “wrapping around” from the bottom layer to the top layer.

Similarly, the logic block marked XL determines the expansion status of the node below the current node. However, since layers are processed from bottom to top, this status must reflect the state of the lower layer *before* the current expansion step. Thus XL is asserted true when the *current* layer is being processed and stored in a flip-flop for use during the next clock cycle, when it reflects the expansion status of the layer below the current layer. As in the XH logic and AND gate connected to /TOP suppresses the XL logic when the top layer is being processed since to keep expansion wrapping around from the top layer to the bottom layer.

The preferential routing input PFV allows vertical expansion to be suspended while allowing horizontal expansion to proceed at an expanded rate. Specifically, when the PFV input is zero the shift register is bypassed and the sequencer’s next state is fed directly back into the ST0 register. This has the effect of performing horizontal expansion on a single layer in isolation from all other layers. Holding PFV zero for a number of clock cycles before asserting it high biases expansion in the horizontal directions.

The PFV input can also be used to reduce the execution time of the backtrace phase – when the TRACE command is applied to a cell and the indicated backtrace direction is horizontal, the control unit can assert PFV to zero so that the current layer is processed again on the next clock cycle rather than waiting L clock cycles for the layer to recirculate through the shift register. Thus successive horizontal backtrace steps take one clock cycle each. A vertical backtrace step in the “up” can also be executed in one clock cycle using the normal action of the shift register.

On the other hand, a vertical backtrace step in the “down” direction will take L-1 cycles since all the other layers must be recirculated through the shift register before the new layer is reached. This exposes a potential problem with the design: In the worst case, where a connection is being made from the top routing layer to the bottom routing layer,  $(L-1)^2$  clock cycles will be required during the backtrace phase. This penalty is acceptable when the number of layers is fairly small, say, less than 10, and top-to-bottom routing connections are rare. When this is not the case, the hardware can be modified to use a bidirectional shift register. This would allow backtrace in the “down” direction to proceed in just one clock cycle but would add considerable cost to the hardware for each cell.

## V. CONTROL UNIT DESIGN

The control unit sequences the operation of the cells in the array. It accepts commands and operands from an external host such as a personal computer and returns status and data. Commands and status are communicated one byte at a time. The current version uses an open source RS-232 serial interface [20] to ease interfacing with the PC host. However, this interface can easily be replaced with a higher performance interface such as a PCI bus interface.

Table 2 summarizes the commands implemented by the control unit

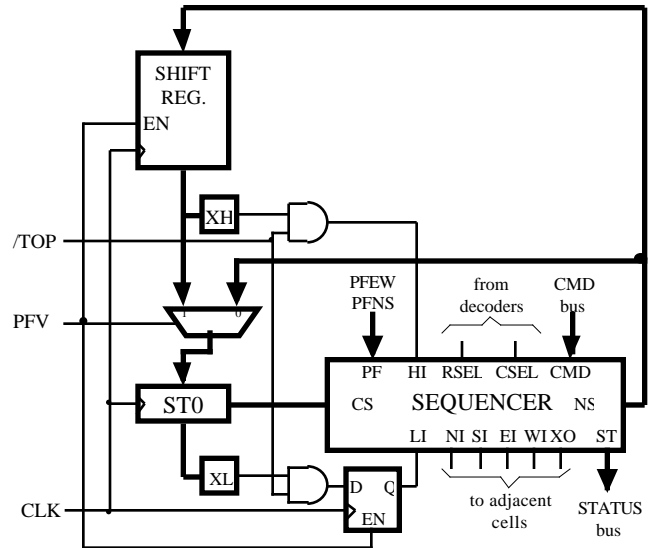


Figure 3. Internal implementation of the multi-layer cell. Layers are processed from bottom to top while the state of each layer is circulated through the shift register.

Table 2 – Commands executed by the control unit.

Command	Parameters
ROUTE	sx, sy, sz, tx, ty, tz
SELECT	x1, y1, z1, x2, y2, z2
CLEAR	(selected region)
CLEARX	(selected region)
EXPAND	(selected region)
SET	(selected region)
TRACE	(selected point)
GET_XCOUNT	(none)
GET_TCOUNT	(none)

The ROUTE command accepts a sequence of coordinates for the source and target of a desired connection. The control unit responds to this command by performing cleanup, expansion and backtrace. If routing is successful, it returns the coordinates of the endpoints of the wire segments used to make the connection followed by a "SUCCESS" status code. If routing is not successful, separate status codes indicate that the failure occurred either in the expansion ("XFAIL") or backtrace ("TFAIL") phases.

The SELECT command selects a (possibly multi-layer) rectangular region of cells to which a primitive cell command will later be applied. These include CLEAR, which sets all selected cells to the E state, CLEARX, which sets only cells in an expanded state (i.e., XE, XW, XN, XS, XU, XD) to the E state. In addition, the EXPAND, SET and TRACE commands apply the corresponding low-level cell commands to the selected cells for debugging purposes.

The GET\_XCOUNT and GET\_TCOUNT commands return the values of cycle counters for the expansion and backtrace phases, respectively. These counters, which are cleared at the beginning of each ROUTE command, are used in performance measurements.

Figure 4 shows the control unit and its connections to the processing array. Datapath elements in the control unit include three up/down counters labeled  $x1$ ,  $y1$ ,  $z1$  and three registers  $x2$ ,  $y2$ , and  $z2$ . During routing, the  $(x1,y1,z1)$  counters are loaded with the target coordinate of the desired connection while  $(x2,y2,z2)$  registers are loaded with the source coordinate. The control FSM applies the SET command to set the source to the XE state and then applies the EXPAND command over multiple clock cycles while selecting the target coordinate and monitoring the status bus. If expansion completes successfully, the cell representing the target will pull status signal S1 low when it is reached. The control unit then performs the backtrace phase using the  $(x1,y1,z1)$  up/down counters. More specifically, it applies the TRACE command to the cell selected by  $(x1,y1,z1)$ , reads the backtrace direction of this cell from the status bus, and increments and/or decrements the appropriate up/down counters to select an adjacent cell in the direction of the shortest path. This process continues until the  $(x1,y1,z1)$  coordinate matches the source coordinate stored in the  $(x2,y2,z2)$  registers, as detected by the three comparators in the datapath.

The layer selection logic is used to determine when a command should be applied to a specific layer or range of layers. This is used to apply commands to a rectangular region of cells over a range of layers, for example, when clearing a range of cells to rip up a net.

The multiplexer at the bottom of Figure 4 is used to select which information will be passed back to the host PC. This includes the endpoints of segments that are routed in the backtrace phase (from the  $x1$ ,  $y$ , and  $z1$  registers), the status output of the cell array (used for debugging), status codes that indicate error conditions and/or successful completion, and cycle counters that are used for performance measurements.

The datapath elements of the Control Unit are sequenced by a Finite State Machine which reads commands from the host interface, sequences the various control signals, and reads status signals from the datapath and the processing array. Currently it contains 22 states.

## VI. RESULTS

### A. Implementation

The design of L3M accelerator has been coded in Verilog HDL and synthesized using the Xilinx ISE 4.2i and Synopsys FPGA Express tools targeting a Xilinx XC2S300E FPGA [18] on a Memec Development Board [19]. This board includes a 24Mhz clock so synthesis was performed with a 41ns timing constraint.

When implemented alone, the synthesized multilayer cell required between 27 4-input look-up tables (LUTs) when preferential routing was disabled up to 32 LUTs when preferential routing is included. Three of these LUTs implement shift registers using the Xilinx SRL16 feature; each cell also requires three flip-flops and three tristate buffers for the STATUS outputs.

The LUT requirements of the L3M cells can be used to predict the maximum size of routing array that can be

accommodated by a larger FPGA. For example, the Virtex-II Pro XC2VP125 device [18] contains 111,232 LUTs and could accommodate a 58 X 58 array of L3M cells.

Table 3 shows the results of synthesizing the complete L3M multilayer router for array sizes 4 X 4 X 4 with 8 X 8 X 4 with no preference inputs used and 8 X 8 X 4 with vertical preferences used to speed up the backtrace phase, as described at the end of Section 4. The first two versions easily met the 41ns clock timing constraint, while the version using vertical preference requires a slower clock of 45ns.

Lookup-Table (LUT) and Flip-Flop (FF) measurements in Table 3 reflect the implementation cost. All three versions fit comfortably in the XC2S300E device and leave room for implementing the serial interface, which requires an additional 173 LUTs and 141 flip-flops.

Table 3: Implementation Costs and Predicted Clock Period.

Array Size	Control LUTs/FFs	Array LUTs/FFs	Total LUTs/FFs	CL (ns)
4 X 4 X 4	238 / 23	425 / 50	663 / 73	37
8 X 8 X 4	244 / 27	1819 / 194	2063 / 221	38
8 X 8 X 4 VP	386 / 43	1841 / 259	2227 / 302	45

### B. Performance

To compare the implementation of the hardware accelerator with software, a reference software implementation of the Lee Algorithm was written in ANSI C. In this program the grid is represented by a three dimensional array of bytes, with each byte representing the state of one gridpoint. This software implementation supports just two-terminal routing and no preferential routing. The resulting code consists of about 1,000 lines of C and was compiled using gcc version 2.96 with the "-O2" optimization option. Performance measurements were made on a 2.53GHz Pentium 4 system with 1 GByte of RAM running Redhat Linux version 7.3.

To measure software performance the code was modified to access the Pentium 4 cycle counter using code provided in [21]. This provides a measurement of the number of clock cycles needed to execute each expansion, backtrace, and cleanup step.

Similar measurements of hardware performance were made using the cycle counters designed into the L3M Control Unit, as described in the previous section. Measurements were made for two versions of the hardware router.

The first version makes no use of the vertical preference hardware. It requires less hardware and easily meets the 24MHz clock target for the development board. However, the backtrace phase must wait several clock cycles on each step for the right layer. This version easily meets the target clock period of 41 ns.

The second version uses the vertical preference hardware to reduce the number of clock cycles required during backtrace – as described at the end of Section IV. This version does not meet the 41 ns timing constraint – the Xilinx place and route software predicts instead a minimum clock period of 45 ns (22.2 MHz). Measurements for each hardware version were made using a Verilog simulator.

Clock cycle measurements were made for both software and hardware versions by routing a sequence of ten randomly selected source/target pairs on an initially empty grid of size 8 X 8 X 4. Figure 5 shows the locations of the source and target terminals of each pair along with the route found by the hardware router.

Table 4 summarizes for each implementation the total clock cycle measurements, the equivalent execution time (i.e. clock cycles \* clock period), and the speedup of each hardware implementation compared to the software implementation.

Not surprisingly, the biggest source of speedup comes from the expansion phase. There are two reasons for this: first, the parallel expansion of multiple gridpoints in the hardware array provides a significant speed up; and second, even the cost of expanding a single gridpoint in software averages about 4,000 cycles. Therefore the hardware implementation pays off even for short routes with relatively little parallelism.

Given the speedup of the expansion stage, the amount of time consumed by the backtrace stage becomes significant, and in the first hardware router version consumes more clock cycles than expansion and cleanup combined. Using the vertical preference feature during backtrace significantly speeds up routing even though the overall clock period is increased and the expansion and cleanup phases are proportionally slower.

## VII. CONCLUSION

This paper has described a new architecture for multilayer maze routing and its implementation in a low-end FPGA for an 8 X 8 X 4 routing grid. Preliminary measurements show the promise of this approach, with speedups over software ranging from 50-75 depending on the specifics of the hardware which were used. Larger versions of the accelerator will offer increased speedups since the amount of parallelism in expansion will be increased even further.

There are many directions for future work in this project. First of all, the development of larger routing arrays is currently underway using a Xilinx XC2V6000 FPGA. This should allow the design of arrays at least 32 X 32 in size, which is more appropriate for VLSI routing using a global routing/detailed routing paradigm. Second, the logic design of the architecture can be tuned to reduce propagation delays and so increase the clock rate. Third, host interface needs to be developed that supports routing of multiple nets while performing rip-up-and-reroute. Finally, the architecture needs to be extended to support more advanced features like variable width and spacing requirements.

## ACKNOWLEDGMENTS

This author acknowledges the assistance of C. Briand, T. Ihimoyan, and M. Marbell in implementing early versions of the L3 hardware and software and helpful discussions with M. Abramovici and J. Greco.

## REFERENCES

- [1] Sarrafzadeh, M. and Wong, *An Introduction to VLSI Physical Design*, McGraw-Hill, 1996.
- [2] Lee, C. Y. "An Algorithm for Path Connections and its Applications," *IRE Transactions on Electronic Computers* vol. EC-10, no. 2, pp. 346-365, 1961.
- [3] Breuer, M., and Shamsa, K. "A Hardware Router," *Journal of Digital Systems*, vol. IV, no. 4, pp. 393-408, 1981.
- [4] Iosupovici, A., "A Class of Array Architectures for Hardware Grid Routers", *IEEE Trans. CAD*, vol. CAD-5, No. 2, April 1986.
- [5] Ryan, T. and Rogers, E., "An ISMA Lee Router Accelerator", *IEEE Design & Test of Computers*, pp. 38-45, October 1987.
- [6] Suzuki, K., et. al., "A Hardware Maze Router with Application to Interactive Rip-Up and Reroute," *IEEE Trans. CAD*, vol. CAD-5, no. 4, pp. 466-476, June 1986.
- [7] Watanabe, T., et. al., "A Parallel Adaptable Routing Algorithm and its Implementation on a Two-Dimensional Array Processor", *IEEE Trans. CAD*, vol. CAD-6, no. 2, 1987.
- [8] Ventkateswaran, R., and Mazumder, P., "Coprocessor Design for Multilayer Surface-Mounted PCB Routing," *IEEE Trans. VLSI Systems*, vol. 1, no. 1, 1993.
- [9] Rutenbar, R. and Mudge, T., "A Class of Cellular Architectures to Support Physical Design Automation", *IEEE Trans. CAD*, Vol. CAD-4, No. 4. October 1984.
- [10] Won, Y, Sahni, S., and El-Ziq, Y., "A Hardware Accelerator for Maze Routing", *Proceedings Design Automation Conference*, June 1987.
- [11] Banerjee, P., *Parallel Algorithms for VLSI CAD*, Prentice-Hall, 1994.
- [12] McMurchie, L., and Ebeling, C., "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs", *Proceedings International Symposium on FPGAs*, Feb. 1995, pp 111-117.
- [13] Chan, P. and Schlag, M., "Acceleration of an FPGA Router", *Proc. 5<sup>th</sup> Annual Symposium on FPGAs for Custom Computing Machines*, April 1997.
- [14] DeHon, A. Huang, R. and Wawrzynek, J., "Hardware Assisted Fast Routing", *Proc. IEEE Symposium on Field-Programmable Custom Computing Machines*, April 2002.
- [15] Huang, R, Wawrzynek, J., and DeHon, A., "Stochastic, Spatial Routing for Hypergraphs, Trees, and Meshes", *Proc. International Symposium on FPGAs*, February 2003.
- [16] Nestor, J., "A New Look at Hardware Maze Routing", *Proceedings Great Lakes Symposium on VLSI*, March 2002.
- [17] Nestor, J., "FPGA Implementation of a Maze Routing Accelerator", *Proceedings International Conference on Field-Programmable Logic and Applications*, Sept., 2003.

[18] Xilinx, Inc., Xilinx Databook, 2003. Available online at <http://www.xilinx.com>.  
 [19] Memec, Inc., *Spartan-II Development Board User's Guide*, 2002.

[20] R. Usselmann, "Simple Asynchronous Serial Controller" Available at <http://www.opencores.org/projects/sasc>.  
 [21] R. Bryant and D. O'Halloran, *Computer Systems: A Programmer's Perspective*, Prentice-Hall, 2002.

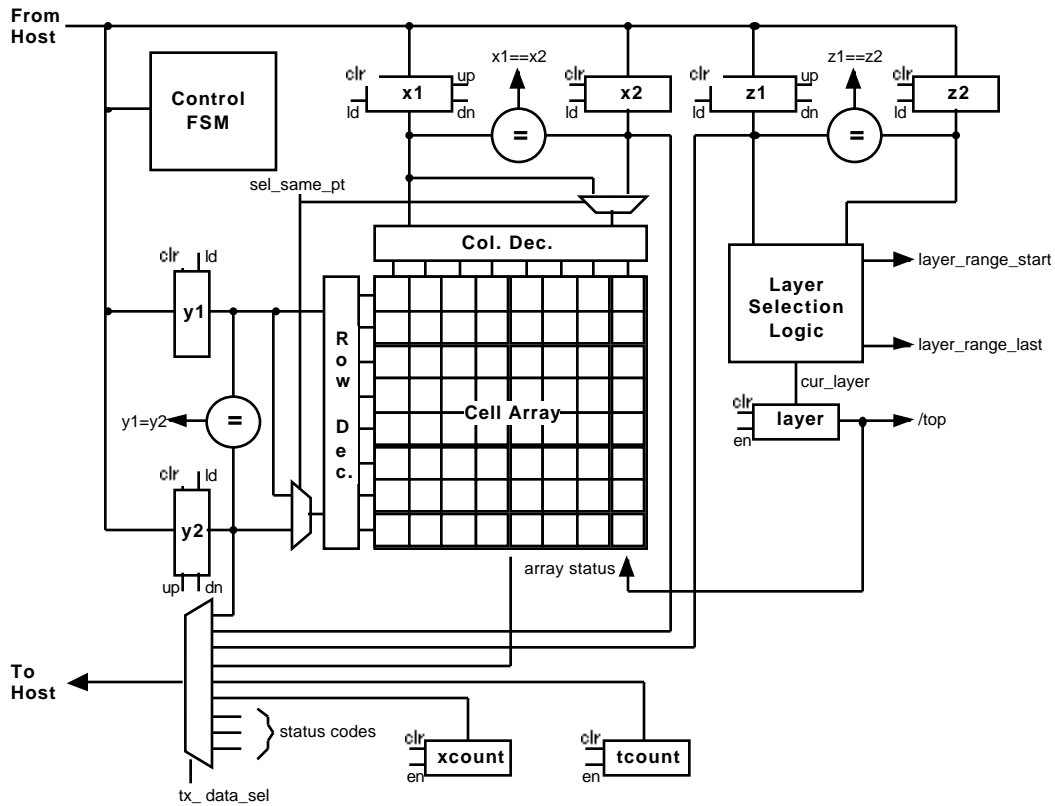


Figure 4 – Connections between Control Unit and PE Array

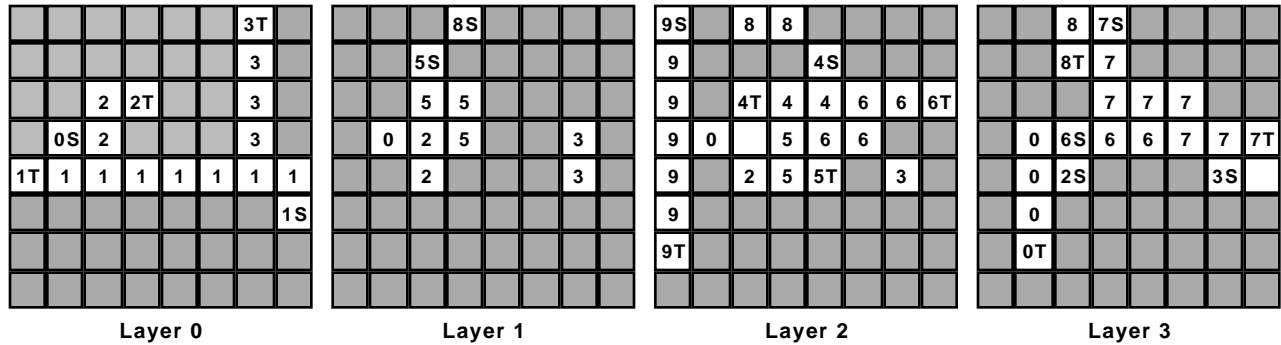


Figure 5 – Result of routing ten random connections on an 8 X 8 X 4 accelerator.

Table 4 – Performance summary for routing ten random connections.

Phase	Software (2.53 GHz)		L3M (24MHz)			L3M w/ mod. Backtrace (22.2Mhz)		
	Cycles	Time (µs)	Cycles	Time (µs)	Speedup	Cycles	Time (µs)	Speedup
Cleanup	97,512	38.54	40	1.64	23.50	40	1.80	21.41
Expansion	2,860,280	1130.55	227	9.31	121.47	227	10.22	110.68
Backtrace	78,084	30.86	319	13.08	2.36	82	3.69	8.36
Total	3,035,876	1199.95	586	24.03	49.94	349	15.71	76.41